Software Technology 1

C サンプル&ツール集

トレーニングから実践ツールまで

C言語研究会 編

技術評論社

Software Technology 1

Cサンプル&ツール集

トレーニングから実践ツールまで

C言語研究会編

技術評論社

Cサンプル&ツール集

C言語研究会編

技術評論社

序

C言語は，今後に可能性を秘めたコンピュータ言語として，広く知られるようになってきました．この言語の特長のひとつは，同じ高級言語であるPASCALなどに比べると，よりハードウェアに即した低レベルの記述ができるということです．つまり従来ではアセンブラでなければ書けないといわれていた，機械に密着した操作を可能にしています．その例としては，C言語によってUNIXというOSが作成されたことがあげられます．

C言語はアセンブラに比べた場合に，プログラムの構造化がしやすく，高い移植性を有しています．移植性という面から見た場合，アセンブラはどうしてもCPUに依存してしまう傾向があるのに対し，C言語はコンパイラが作りやすく，CPUを選ばずに他のOSへ比較的簡単に移植を行うことができるからです．

このような優れた特長を持つC言語は，近年とみに注目を集めるようになってきました．ところが，C言語を使ったプログラムのソースは，小さなサンプルプログラムはあっても，実用に耐えうるツールやユーティリティとなると未だ公開されていないのが実状です．例外としてUNIXのソースがありますが，一般には見ることが困難です．

本書は，C言語の入門書をひと通り読まれた方やこれからC言語を学ぼうとする方のために，本格的なプログラムを公開する目的で書かれたものです．

「第1部 パソコンC言語入門編」では，UNIXの開発に伴い産み出されたC言語をパソコン（OS）上で使った場合の作業環境の実際や，各OSおよび各CPUとC言語との相性や使い勝手を解説するほか，各処理系別のベンチマークテストのデータを公開しています．

「第2部 Cトレーニングサンプル編」では，実際にツールを使う前のトレーニングとして，主要なコマンドや関数がプログラム上でいかに使われているのかを示し，初歩のCプログラミングテクニックを紹介しました．

「第3部 Cツール編」では，CP/M-80, MS-DOS, OS-9 別に実用ツールを公開しています．いくつかのツールには，プログラム中に子細な注釈を記し，アルゴリズムを把握する一助になればと考えました．

最後に，本書の上梓に際して，第1部の原稿を寄せて戴いた小幡広昭氏およびプログラム作成に日夜邁進して下さったソフトウェアスタッフ諸氏に謹んで感謝の意を表します．また，ソフトを提供して戴いたソフトウェア・インターナショナル(株)，マイクロソフトウェア(株)，(株)サザン・パシフィックおよびデイジーホイールプリンタを提供して戴いたブラザー販売(株)の方々に紙上を借りてお礼を申し上げます．

1984年　9月　　C言語研究会

もくじ

第1部　パソコンC言語入門編

第2部 Cトレーニングサンプル編

第3部　Cツール編

各処理系の関数一覧表

第1部

パソコンC言語入門編

UNIXとCの作業環境

小幡広昭

UNIXのツール

UNIXというOSを説明するときに，他のOSと比べて使い勝手がよいということが強調されます．その理由の1つは，UNIXは豊富なツール類が機能的に分けられていて，組み合わせて使えるように設計されているからです．

UNIXでは単純な機能を持つものを結合して目的を達成しようという考え方があり，ごく単純なCの関数からファイル・システムやシステム・コールといった高度なものに至るまで，この考え方は貫かれています．しかも，それぞれの機能は同レベルの間で横方向にも，縦方向にも組み合わせやすいように細かな配慮がなされています．

このようにソフトウェアを何かの目的を達成するための道具としてとらえるという考え方から，UNIXではツールという言葉が使われます．出来合いのツールを活用し，他の人にも使ってもらえるようなツールを設計することにより，目的を達成してきたわけです．UNIXに今備わっているコマンドや機能の多くは，あらかじめ用意されていたのではなく，UNIX自身を作り上げていく過程や，UNIXを使って何か仕事をしようとする過程で，ツールという考え方に基づいて追加や修正，選択されてきたものです．つまりUNIXのソフトウェアは，UNIX自身を使って作成され保守されてきたのです．

UNIXのソフトウェアはシステム部も含め大部分がCで記述されています．そのためUNIXを使ってのプログラム開発能力は，現在のUNIX自身のプログラムやUNIX上で動くプログラムを作るくらいは，充分であるということがいえます．

UNIXは，このように成長していくことを前提として設計されているので，新しいコマンドや機能を容易に受け入れられるように作られています．つまりUNIXを使う人は，様々なレベルで自分なりのツールを作り，システムを作り上げていくことができるのです．

ツールとして役に立つプログラムの書き方や，その設計と実現の方法を具体的なプログラム例を挙げて説明した本があります．大変参考になると思いますすので，ぜひ一度読んでみてください．

B.W.Kernigham, P.J.Plauger「Software Tools」: Addison-Wesley(1976)
邦訳　木村泉訳「ソフトウェア作法」: 共立出版(1981)

UNIXのツールとC言語

UNIXの多くのツールはCと密接な関係にあります．UNIXのツールのうちで特定のプログラミング言語のための開発ツールは，C言語のためのものがほとんどを占めています．これらのツールには，Cのプログラム自身を作るためのものや，コンパイル，リンク，エディット，デバッグ等のための便利なツールがたくさん用意されています．また，ツールを動作させるために一種のプログラムのようなものが必要な場合には，その多くがCと似た形式で与えられます．またUNIXのツールは，ほとんどすべてがCで書かれていますので，機能を拡張したり，他のプログラムの参考にするためには，どうしてもCの理解が必要となります．

このように，UNIXシステムでは，Cでのプログラミングが前提と思われるような構成がとられています．Cでプログラミングをしていくことにより，UNIXの便利なツール類を使いこなせるようになれば，UNIXをさらに深く理解することができるようになります．そのためには,UNIXのソース自身がとても参考になるのですが，残念ながらソース契約者にのみ公開されるため，一般の人にとっては機会は非常に限られています．後に紹介されているプログラム集が，少しでも役に立てばと思います．

パーソナル・コンピュータとUNIX

UNIXのコマンドの多くは，UNIX独自の機能を生かして働くように作られています．たとえば，フィルタと呼ばれるものがそれです．これはパイプライン機能を使うことによって，はじめてすっきりとした概念となります．UNIXにはパイプ機能の他にも，階層構造のファイル・ディレクトリ，標準入出力によるリダイレクション，シェルとシェル・スクリプトといった，今までのパーソナル・コンピュータのOSにはなかった便利な機能があります．

そのようなことから最近のパーソナル・コンピュータのOSは,UNIXの便利さの恩恵にあずかろうと，部分的にその機能を取り込んでUNIX-likeにする傾向があります．これらには，最初からUNIX-likeとして登場したものや，最初は考えに入れていただけで，バージョン・アップの時点でその機能を取り入れていったもの等，様々です．完全なUNIXを今の16ビットのマイクロ・コンピュータで実現するのは，CPUの性能や周辺装置の能力から，なかなか難しい

のですが，NS16032，M68000，Z8000，i8086等のCPUを使った製品では，次々と発表されてきています．

現在，能力の低いシステムでUNIXを動作させるのに問題となっている主な内容は，CPUの処理速度，特にCPUのアーキテクチャで高級言語に関する部分が弱いこと，メモリ・マネージメント機構と仮想メモリの取り扱い，そして安価で信頼性の高い大容量のディスクとその実効データ転送速度等です．これらの問題が，ある程度解決されればUNIX程度のOSがわりあい簡単にパーソナル・レベルで使えるようになると思われます．

シェル

シェルもUNIXのツールの1つで，ログインしている間は，ターミナルごとに必ず1つは走っているプロセス（プログラム）です．シェルは，与えられるコマンドによってプロセスを生成するときに，ファイルのオープンやプロセス間のパイプの生成等をあらかじめ行います．これがI/Oのリダイレクションやパイプと呼ばれるシェルの機能です．さらにシェルには，シェル・コマンドの手続きをファイルに書いておいて実行させる，シェル・スクリプト機能があります．また特殊文字を使って，タイプの手間を省いたり，パターンに合う名前を選び出したりすることができ，ユーザに快適な作業環境を提供してくれます．

シェル自身，1つのプロセスですから，ユーザが自由に取り替えたり，作り替えたりすることができます．たとえば，Bourne-シェル（sh）より強力なC-シェル（csh）は，カリフォルニア大学バークレー校で作られたものですが，このシェルには，ヒストリ（入力コマンドの履歴リスト），エイリアス（alias：コマンドの別名付け機能），ジョブの途中での停止／新しいジョブの起動／バックグラウンドとフォアグラウンドの移動／ジョブの再開などの機能，シェル・スクリプトについての構文の強化等，多くの機能追加がなされています．これらの機能の多くは，ユーザに，より快適な作業環境を提供するために付け加えられたものです．

プログラム・ジェネレータ

UNIXには単純な機能のツールもたくさんありますが，ここでは，かなり高級なツールであるyacc，lex，awkについて説明しましょう．

yaccという名前は，「もう1つのコンパイラ・コンパイラ」という意味の頭文字をとったものです．つまりUNIXには，以前にすでにコンパイラ・コンパイラがあったわけで，このツールはその後に作られたものなのです．しかしな

がら，現在コンパイラ・コンパイラと呼んだ場合，それはyaccのことを指します．本来のコンパイラ・コンパイラに取って代ったわけです．従って現在では「もう1つの……」という名前はあまり意味がなくなってしまいましたが，UNIXは前にも述べたとおり，プログラムの作成者が適当な名前を付けて追加，成長してきたので，コマンド名にはあまりこだわる必要はありません．ちなみに，この後で説明するawkは，三人の作成者の頭文字からできています．

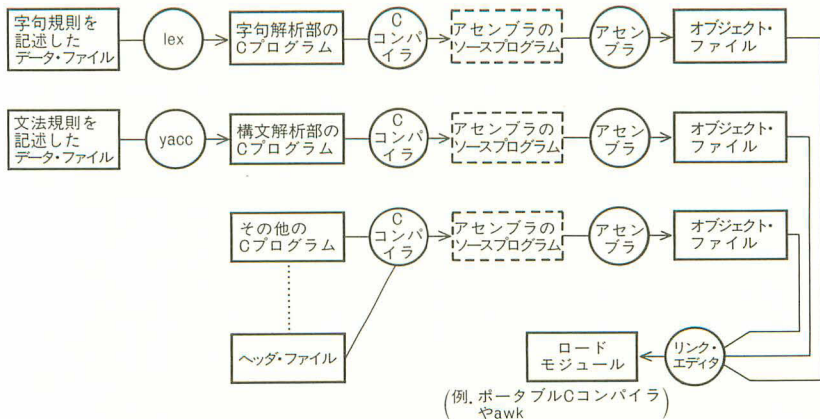
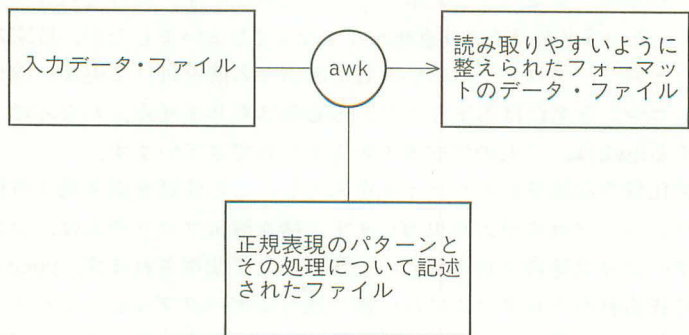
yaccは，言語仕様の記述されたファイルを入力し，この仕様を満す構文解析のためのCのソース・プログラムを出力します．構文解析プログラムは，コンパイラやインタープリタ等のプログラムの一部分として使用されます．yaccを使って今までに作られたプログラムには，後で述べるポータブルCコンパイラの他に，APL，Pascal，Ratfor，電卓言語，簡易言語，文書検索システム，Fortranデバッキング・システム等，たくさんあります．

lexはyaccと同じように，字句解析部のCプログラムを生成します．コンパイラの字句解析部での表現の認識は，lexによって生成される決定性有限オートマトンによって行われます．

yaccとlexはかなり高級なツールですが，UNIX上で実際に動かしながら学ぶことができるので，コンパイラ等を作る時の参考となります．yaccとlexを使ったプログラムの1つにawkがあります．awkはパターン走査処理言語で，正規表現で指定されるパターンに対して処理する指示を，Cに似た制御構造で与えることにより，データを処理するプログラムです．awkはその用途から，レポート・ジェネレータとも呼ばれています．

図1　yaccとlexを使ったプログラムの作成例

図2　awkの使われ方

入力データ・ファイル
awk
読み取りやすいように
整えられたフォーマッ
トのデータ・ファイル
正規表現のパターンと
その処理について記述
されたファイル

開発言語としてのC

小幡広昭

UNIX上で，Cで記述するクロス・ソフトウェアの開発を行う場合には，今までに説明したツールの他にも便利なものがたくさんあります．ここでは，lint，make，sdb，viについて説明します．

Cプログラムのチェックを行う　lint

lintは，Cのソース・プログラムを調べて，バグや不明瞭な点を検出するツールです．lintは，Cコンパイラよりも厳密にデータの型のチェックを行い，異なるマシンやオペレーティング・システム間の移植上の問題点を洗い出します．さらにオプションでは，厳密にいえば合法ではあるが，無駄が多かったり，エラーを発生しやすい各種の構造も検出します．また，ライブラリについては，一貫性のチェックが行えます．

コンパイラとは別のlintが文法的なチェックを代行するため，Cコンパイラはその部分を省略することができ，高速にコンパイルを実行することができます．

プログラム保守に便利な　make

一般に，ある程度大きなプログラムを作る場合は，いくつかのモジュールに分けて作成し，別々にコンパイルして，その後リンクするという方法が取られます．この時，ソースの一部を変更した場合に，どのファイルを再処理，または再コンパイルすればよいのか分からなくなってしまうことがあります．このような時にmakeが大変役に立ちます．

また，1つのプログラムを小さなプログラム単位のファイルに分けた場合，あるものには全く別の処理が必要なこともあります．たとえば，図1のようにプログラムの一部をプログラム・ジェネレータであるyaccやlexで生成することもあります．また，小さなプログラムのいくつかは，アセンブラで記述されているかも知れません．

makeは，このようなファイルをどのように処理していって最終的なプログラムに作り上げればよいのかという情報を，あらかじめ記述されているファイルから得て，最小限必要な処理だけを行ってプログラムを作り上げてしまいます．

たとえば，ヘッダ・ファイルを修正した場合，そのヘッダ・ファイルを使用しているすべてのファイルを再コンパイルしてくれます．また，いくつかのファイルをいろいろな過程で修正していった場合にも，makeは結果的に影響のあるすべての処理を，忘れずに，しかも誤りなく実行してくれます．

makeは，何らかの処理の結果として作られるファイルの作成された日時と，それを作るのに必要なファイルの最終修正日時を比べて処理が必要かどうかの判断をします．makeコマンドを使うには，ファイル相互の従属関係と処理するのに必要なコマンド列に関する情報を，あらかじめファイルに入れておく必要があります．

Cプログラムのデバッガ　sdb

Cのプログラムのデバッグに際して必要なのが，sdbシンボル・デバッガです(バークレー版4.2bsdではdbxになっています)．sdbは，現在のところVAX-11や68000等のシステムの一部でしか動作していませんが，マイクロコンピュータのシステムでUNIXが使われることが増えるにしたがい，その必要性から動くものが増えてくると思われます．

sdbは，Cプログラムの暴走や違法命令の実行等によって作られるコア・イメージから，ソース・レベルによる関数のトレースや，プログラム・テストのためのブレークポイントの設定と実行，シングル・ステップの実行，変数内容の確認や変更等のデバッグ作業が行えます．

強力なスクリーン・エディタ　vi

通常，ソース・プログラムの作成や修正には，エディタが用いられます．viはスクリーン・エディタですが，/etc/termcapというファイルにターミナルの機能を登録しておくことにより，様々な種類のターミナルで使用することができます．

たとえばシリアルI/Oポートを持つマイクロコンピュータ自身を，UNIXシステムのターミナルのかわりに使用することもできます．しかも画面の文字数は80×24でなくてもかまいません．この情報も/etc/termcapに入れておけばいいのです．

viというエディタは，実はexというラインエディタの編集モードの1つとなっています．ですから，viとexはファイルを編集中に自由に行き来ができます．viよりexのコマンドを使った方がより簡単にできるこも多くあるため，経験の豊かなユーザは，viとexのコマンド・モードをうまく使い分けています．

viの多くの機能の中でCプログラムの編集に特に便利なものとしては，自動字下げ，行全体の左右への一定量の移動，カッコの左右の対応の確認，関数の宣言位置までのカーソル移動等があります．

ポータブルCコンパイラと移植性

UNIX上の強力なツールを駆使し，ある程度のクロス・ソフトウェア開発が終わった時点で開発用のUNIXを完全に切り離し，それ以降の開発はターゲット・マシンのシステムで行うことがよくあります．この場合，UNIX上のクロス・コンパイラ，クロス・アセンブラ，そしてリンク・エディタをCで記述しておけば，ターゲット・マシンにも，ある程度容易に移植することができます．

また，このようにすると，その後はどちらのマシンでもCプログラムのロード・モジュールを生成することができるようになります．

そのためにはUNIXのツールの1つであるポータブルCコンパイラを使用します．ポータブルCコンパイラはそれほど効率の良いコードは生成できませんが，いくつかのメリットがあります．ポータブルCコンパイラは，UNIX上で通常使われているCコンパイラのバージョンと高度な互換性があります．

また，コンパイラ全体の約75%がマシンに独立した部分で，残りの約25%はマシンに依存した部分です．この25%の部分を作りかえれば簡単に移植が行えます．ただし，マシンに依存した部分の多くは単純な変換で済みますが，コードのある一定量は，変換するのが非常に難かしく，かなりの知的努力を必要とします．

ポータブルCコンパイラは，2パスに分割されています．第一パスが構文解析（パーサ）で，第二パスがコードの生成を行います．出力されるのはアセンブラのソース・プログラムです．第一パスの構文解析部には，yaccによって作られたCプログラムが使用されます．また，ポータブルCコンパイラはlintプログラムの主要部分としても使用されています．

このように，異なるシステム間に高度に互換性のあるCコンパイラがあれば，必要なシステム・コールとライブラリ関数を用意することによって，Cで記述されたプログラムの移植性は非常に良くなります．従って，プログラムの移植性を考えた場合には，コンパイラはなるべくサブセットでなく，しかもライブラリ関数の仕様等も含めて，なるべく互換性のあるものが望ましいといえます．

Cとアセンブラ

Cでプログラムを記述する場合，アセンブラを使う必要はほとんどありません．特にUNIX上で実行させるプログラムに関しては，特殊なものを除いて，すべてがCだけで記述できます．

これに対して，ターゲット・マシンがOSもなにもない環境であったり，システムがCに対してサポートしていない場合には，プログラムの一部をアセンブラで記述する必要があります．たとえば，ソフトウェア割り込みのような特殊命令を使う必要がある場合や，ハードウェア割り込み処理部，さらにCPUのアーキテクチャによっては入出力命令等はアセンブラで記述する必要があります．この時，アセンブラで記述する部分を，Cの関数として呼べる形にしておけておけば，他の部分は完全にCだけで記述することができます．

ただし，少しでも処理速度を上げたい場合や，できる限りプログラム容量を小さくしたいというようなときには，アセンブラを使わなければならないこともあります．しかしアセンブラで記述するのに比べ，Cコンパイラが生成するコードの量と質は，他の高級言語ほど大きくはなく，効率もそれほど悪くはありません．特に，今まで数値計算と制御を高級言語とアセンブラの組合せで行っていたような分野では，Cだけで大部分を記述できるようになります．

マイクロコンピュータとC

マイクロコンピュータの分野では，短期間のうちに，次々と新しいCPUや周辺装置が使えるようになってきました．そのため，ソフトウェアについては，大幅な修正なしに新しいマシンにも移行できる必要性が高まってきています．

また，開発終了後のことを考えてみた場合，ハードウェアの変更やソフトウェアの仕様変更に伴う適応性と保守の容易さも，もちろん要求されます．

このような条件を満すプログラミング言語としては，CのほかにもPascal，Praxis，Simula，PL/IそしてAda等があるわけですが，現在のマイクロコンピュータの能力とUNIXというOSの開発環境を考えると，Cが最も適しているように思われます．

Cの特徴と注意点

Cの特徴である柔軟性や簡潔性，そして移植性等は，プログラミングをする人がそれを生かすように，常に心がけていなければなりません．というのは，Cのこのようにすぐれた特徴は，使い方によってはCの欠点となってしまう恐

れもあるからです．

Cはアセンブラのように無防備ではありませんが，どちらかというと，すべてに寛大な言語です．つまり，プログラムの作り方次第で，かなりのことができるかわりに危険も伴うということです．また，書き方によっては非常に理解しにくくもなります．ですからCでプログラミングをする時は，その特徴を理解した上で，良いところを生かすように常に気をつけることが大切です．

各CPUとCの相性

小幡広昭　　C言語研究会

最近では，数多くのマイクロコンピュータのマシン上で，Cのシステムが使えます．ここではいくつかのCPUを取り上げ，Cとの相性についての特徴や問題点を見てみることにします．

Z80

Z80では，Cとの相性で問題となる点がいくつかあります．

第一に，整数のビット長と，それを扱うCPUの命令の効率が悪いということがあげられます．C言語では，個々のデータ型について，その長さを明確に決めてしまうような仕様はありません．たとえば整数のint型は，マシンにとって最も自然な整数のビット長が使われます．shortやlongについても同じことがいえます．

これは，コンパイラによって出力されるコードの実行効率を，そのマシンで最も高めることができるように，という考えからです．そのため，一般的には，int型はそのマシンのアドレスやデータを扱うレジスタと同じ長さにしています．

Z80ではアドレスは16ビットであり，他のプログラムの移植を考えた場合にも8ビットでは足りないので，一般に16ビット長の整数が使われます．ところが，Z80の基本演算は8ビット単位を主体に設計されているため，Cコンパイラが整数を扱う時に出力するコードは，16ビットのマシンに比べてかなり複雑になってしまいます．

もう1つZ80で効率を落す原因として，フレーム・ポインタの問題があります．Cの仕様では，Cの関数は再帰的に呼び出すことができます．そのためには，関数が呼び出されるごとに，スタックにフレーミングを行い，引数や関数内で使われる自動変数等のための作業領域を確保します．この時，フレーム内の値を操作するのに必要となるのがフレーム・ポインタです．

Z80では，フレーム・ポインタとして，たとえばBCレジスタ等を使うわけですが，フレーム内の値を扱うための効率は，かなり悪くなります．特にCでは，文字単位の入出力等のように，関数が頻繁に呼び出されるので，実行時の効率にかなりの影響を与えることになります．

以上の2つの問題に加え，CPUの処理速度が遅いとか，アドレッシング・モ

ードが少ない等のCPU性能上の問題もあるため，Cコンパイラの中には8ビットのCPU独特の工夫をしているものもあります．たとえば，オプションの指定で，再帰的呼び出しのできない形でコードを生成して効率を上げたり(LSI-C等)，char型どうしの演算はint型に変換せずに行う等の処理をしています．これらは，組込型の用途では効率を重視する立場からよいのですが，コンパイラの移植性を悪くすることも確かです．

Cコンパイラをサブセットにすることも，CPU性能の問題から行われることだと考えられますが，これも当然コンパイラの互換性を下げることになります．

もともと，C言語の仕様はコンパクトにまとまっていて，サブセットについての規格といったものはありません．従って，Cコンパイラでサブセットといわれているものの仕様は，それぞれのコンパイラで異なります．このようなサブセットのコンパイラは，今までアセンブラで記述していたものを，なるべく実行効率を下げずに開発効率を上げたい，という場合には有効です．サブセットはコンパイルやリンクに要する時間がそれだけ短く，生成されるコードもフルセットのものに比べて効率が良いか，最悪でも同じだからです．

このように，Z80のような8ビット系のCPUでは，Cの仕様を完全に満すことと，効率を重視することの両立が難しいので，使用目的と将来の方針を明確にして選択する必要があります．

8086

8086は，CPUのレジスタの長さが16ビットであり，BP(ベース・ポインタ)レジスタがフレーム・ポインタとして使用されるのに適しているため，8ビット系のCPUのマシンに比べ生成コードの効率が良く，実行効率も向上しています．一般に，8086用のCコンパイラはフルセットのものが多く，そういった意味からも実用的になってきています．

また，最近の8086用Cコンパイラのほとんどが，浮動小数点演算のためのコプロセッサである8087をサポートしているので，数値計算を行う分野では，かなり強力になっています．

しかし，CPUのアーキテクチャの要請で，メモリの使用にはセグメント・レジスタを扱わなければならず，メモリ容量が64Kバイト以上か以下かで実行効率に影響があるため，生成するコードを変える必要があります．しかも，この選択は一般に，コンパイルやリンク時にユーザが指定する方法が取られます．

この指定はメモリ・モデルと呼ばれ，スモール，コンパクト，ミドル，ラージ等がありますが，そのモデルの種類の数はコンパイラによって違います．

また，ライブラリ関数も，それぞれのメモリ・モデルに対応したものが用意されているのが普通です．ユーザがアセンブラで関数を用意する場合も，引数の受け渡し等はメモリ・モデルごとに異なることがあります．

Cで記述したプログラム中で，ポインタの扱い等を誤った場合，とんでもないアドレスのメモリ内容を破壊したりして暴走させてしまうことがよくあります．また，ヒープ・エリアやスタック・フレームによる領域の侵犯による暴走の可能性もあります．後者の場合は，スタック上のフレーミングで領域を確保する時に毎回チェックを入れることもでき，これをオプションで指定させるコンパイラもありますが，その場合でも，危険と実行効率の低下のどちらかを選択する必要があります．

この問題は，8086のマシンが一般にメモリ・マネージメント機能を持っていないことに原因があります．8086のシステムではマルチ・ユーザ環境のものもあるわけですが，Cのプログラムのデバッグ時はシングル・ユーザで使うことが多くなると思われます．また，初心者によるCプログラムのデバッグは，難しい面もあるかもしれません．

8086用のCコンパイラでは，一般にフレーム・ポインタとしてBPレジスタが使われているということは前にも述べましたが，8086には，残念ながらスタック・フレームを意識した命令は全く用意されていません．従って，スタック・フレーミングを実現するメカニズムに関しては，他の16ビット系CPUに比べ効率が悪くなっています．

もう1つの問題は，8086のCPUレジスタが汎用でなく，ある程度使用目的が決まってしまっていて，しかもその数が少ないことです．Cコンパイラではレジスタ宣言は必ずしも有効でなくてもかまわないのですが，実行効率に大きな影響がある場合が多いものです．ところが，8086用のCコンパイラでは確保できるレジスタの数は多くても3個程度で，レジスタ宣言を無視するコンパイラもかなりあります．

6809

6809が究極の8ビットCPUと呼ばれるのは，そのアーキテクチャが簡潔かつ柔軟なためです．内部レジスタはZ80の半分以下しかなく，アドレス空間も64Kしかありません．しかし，豊富なアドレッシングにより，場合によっては16ビットCPUの8086をもしのぐ小回りの良さを示す場合もあります．

6809には，ポインタとして使用できる16ビットのレジスタが4本あり（うち2本はスタック用)，そのすべてでアキュームレータオフセットを含む様々なオ

フセットアドレッシングが可能な上，オートインクリメント（デクリメント）の機能も持ちあわせています．

こうしたアーキテクチャは，C言語に類似するところもあり，コンパイルしやすい環境だといえましょう．

また，オフセットアドレッシングの産物として，インデックスレジスタを擬似アキュームレータとして使用できる点，デクリメントして0になったところでZフラグが立つことによってカウンタとしても使用できる点があるため，特に整数（16ビット）の加減には便利です．

乗算に関しても2つの8ビットアキュームレータどうしの乗算命令があるために，整数の乗算はZ80の比ではない程，高速に行うことができます．

ところで，関数間の引数の受け渡しは6809の場合，真にシステムスタックを使って行うことが容易に実現できます．6809のシステムスタックは，先に述べたように，インデックスレジスタとしての機能も持ちあわせているためにコンスタントオフセットをすることによって，引数を受けとることができるのです．

しかし，こうした技法は，引数受け渡しのための領域と，関数（一般にサブルーチン）からの戻り番地を記録する部分が同居するため，引数の形式を誤ると，ただそれだけで暴走してしまうことにもなりかねません．

一般にC言語では，このようなミスではエラーを発生しないようになっていますので（プログラマの変則的な引数の与え方をすることを許すため），充分注意する必要があります．

さて，システムスタックの柔軟性については，もう1つ特筆すべきものがあります．それは，インラインパラメータの利用が容易だということです．

たとえば，整変数に定数値を加えるという操作は，次のような技法により，この定数値をプログラム中に置けるのです．

```
        BSR  LAB
        FDB  定数値
LAB     PULS  X
        JSR  _add
```

ここで，_addというランタイムサブルーチンは，Xレジスタをポインタとして引数としています．現にこうした技法は，MICROWARE Cの中で利用されています．

ただ，いずれにせよ，64Kというアドレス空間は，Cコンパイラを動作する

にはあまりにも重荷で，システムを構築する程の大きな作業には向かないと考えられますが，アセンブラを含めたＣ言語の学習用としては便利だと思われます．

68000

68000では，8086に比べＣと相性の良い特徴がいくつかあります．

まず，メモリ空間が線型で16Mバイトもあることです．また，link，unlk命令により，スタックのフレーミングの実現が容易になっています．汎用レジスタの数はデータ・レジスタが８，アドレス・レジスタが８（１つはスタック・ポインタ）の合計16もあり，それぞれ32ビット長で，データ・レジスタは８ビットでも16ビットとしても使えます．

CPUの命令はアドレッシング・モードが豊富で，Ｃコンパイラが生成するコードの効率がよくなります．さらにメモリ・マネージメント機構を68451 MMUによって実現でき，Ｃの実用性はさらに向上します．68000へのＣ言語のインプリメンテーションは，そのアーキテクチャにより容易になっているため，かなり早い時期から多くのシステムで行われてきました．

ただし，浮動小数点演算についてはソフトウェアで行うことになるため，他の16ビットや32ビットのCPUのマシンに比べ，かなり遅くなります．UNIXの本来の使い方である，コンパイラ設計等の計算機科学や文書作成の分野では，浮動小数点の演算はあまり行われませんが，Ｃを数値計算のために使うことも多くなってきているので，使用目的によりこの点を考慮に入れておく必要があると思われます．

今まで説明してきたCPUのマシンに比べると，68000のマシンは規模の大きいものが多く，強力ではありますが，それだけ高価なものが多いようです．

しかし，Ｃ言語の需要の拡大とその普及に伴い，さらに実用的なシステムを求める声が多くなってきています．この点で，現在最も要求に合うのが68000 CPUのマシンであり，近いうちに機能が高くて価格の安いシステムが発表されると予想されます．

16032

UNIXとＣ言語を考えた場合，現在最も注目されているCPUが16032です．その理由はいくつかあります．

その１つは，16032CPU，16082MMU，16081FPUの３チップの組合せにより，メモリ管理と浮動小数点演算を含むシステムを容易に作ることができるという

ことです．しかも，MMUによってデマンド・ページング方式による仮想記憶機構を実現できます．この機構は，実用的な複数ユーザのUNIXシステム等，能力の高いOSを考えた時に非常に有効なものです．また，MMUにはプログラムをデバッグするためのブレークポイントと，プログラムの流れをトレースする機能が含まれており，デバッグ時の効率を上げることができます．

もう1つの理由は，CPUのアーキテクチャの設計がすぐれていることです．

まず，高級言語のサポートを考慮に入れて設定されています．たとえば，スタックのフレーミングについては，ENTERとEXIT命令によって，容易に実現でき，しかもフレーム・ポインタとしてCPUレジスタのFPが用意されています．また，モジュール構造のソフトウェアのサポートも積極的に行われています．

さらに，アドレッシング・モードを伴った命令は対称性を持ち，そのアドレッシング・モードの種類も豊富です．しかも，メモリ対メモリの演算を1命令で実行できるので，コンパイラの設計が容易になり，生成されるコードは非常に高い密度にすることができます．この結果，プログラムの実行も他のCPUと比べ速くなっています．

各OSとCの相性

小幡広昭　　C言語研究会

Cでソフトウェア開発を行う場合，作業環境として最もよいと思われるのは，やはりUNIXということになります．しかし，CP/MやMS-DOS，OS-9等のシステム上でも，Cのプログラムを作って動かすことはできます．

もともと，C言語自身はOSに対する依存度が低く，入出力等に関しても標準ライブラリを利用するわけですから，そのOSに適したプログラムを書くのに問題はありません．つまりそのOSにある機能は充分に生かすことができるのです．ただ，UNIXの便利なツールをそのまま使いたいとか，他のOSで動いているCのプログラムを使いたいという場合には，OSがサポートしている環境の違いによって使えないこともあり得るわけです．

最近のOSはUNIXの機能を一部取り入れることで，外見はUNIXらしくなってきています．特にI/Oのリダイレクションや，パイプ機能は擬似的なものも含め，実現が容易なために多くのOSで取り入れられてきています．

ただ，UNIXとUNIXに似たOSは，Cを使用する作業環境としては完全に異なると考えた方がよいと思われます．もちろん，UNIXスタイルのマシンが，近い将来に普及することが予想されるため，ある程度親しんでおくことの意味はあると思います．

それでは，現在のUNIXがツールとして用意しているものを，他のOSではどのようにして実現しているのかを見てみることにしましょう．まず，他のOSに共通しているものとして，lint，sdb，make，viを取り上げます．もちろん，UNIXにはこれ以外にもgrepのように，小さなツールでありながら使い道がたくさんあり，プログラム作成の時にも非常に有効であるものが豊富にあることを忘れないでください．

lintと同じような文法チェックは，他のOSではCコンパイラに含まれているのが普通です．lintでは厳密な文法チェックはもちろん，必要に応じてそれ以上のチェックも行ってくれますが，他のOSのCコンパイラでは，コンパイラの大きさやコンパイルの速度等も考えに入れなければならないので，厳しいチェックはしていないのが実情のようです．これらのコンパイラの中には，コンパイルの初期の段階で，できるだけ多くのチェックとエラー表示を行ってしまい，なるべくユーザが時間を無駄にしなくて済むように工夫しているものもあるよ

うです．

次に sdb ですが，残念ながらこれをサポートしているOSは少ないようです．

これは現在の主流となっているCPUのアーキテクチャが高級言語をサポートするように設計されていないため，作るのがかなり難かしいということにも原因があるようです．従って，他のOSではアセンブラ・レベルでのデバッグが主体となります．このような方法をとる場合，ユーザはコンパイラがどのようにコードを展開しているかを知る必要があり，初心者にはかなり難しい面もあります．

デバッグに際しては，Cのプログラムがアセンブラに展開された時のソース・リスト，または直接機械語に展開された場合は逆アセンブル・リストが必要となります．

makeは，プログラムを作り上げる時には大変便利なのですが，小さなプログラムを作る場合には無くても済むものです．ただ，Cでのプログラミングのテクニックの1つである，単純な機能ごとに分けて作り，これらを組合せて1つのプログラムとする場合には有用なものです．

makeを動かすためのOSの環境としては，個々のファイルに最終修正日時の情報があることが絶対に必要ですが，それ以外にもディレクトリが階層構造になっていて，しかもディスクの容量がある程度より大きいこと，シェル・スクリプトが使えるシェルが楽に動く程度の機能がOSにあること，そしてコンパイラやリンク・エディタの機能が高いこと等が必要です．そうでないと，makeは動かすことができても，あまり実用的ではなくなってしまいます．

UNIXでは，このmakeがあるおかげで，他人の作ったプログラムや，時間が経過してプログラムの構成を忘れてしまったプログラムを機能変更したり，システム・コマンドを作り直したりすることが簡単にできるのです．ただ，上記のような条件があるため，UNIX以外の小規模OSでは，残念ながらまだ実現されていないようです．

viとexほど多くの機能はなくてもかまいませんが，強力で使い勝手の良いエディタは，プログラムの作成にあたってはかかすことのできないツールです．

このようなツールが有るか無いかで，プログラミングの開発効率は大きく違ってきます．また，プログラム作成以外のユーザ環境においても，エディタが果す役割は重要です．できれば，使用する人は同じなのですから，OSが違っても同じエディタが動いてほしいところですが，OSがサポートする環境の違いなどから難しいのが現状です．

次にCP/M，MS-DOS，OS-9について，Cのプログラミング環境を見てみることにしましょう．

CP/M-80

8ビット系のCP/M-80上で動くCコンパイラは，ほとんどがCのサブセットで，なるべくオブジェクトの効率を落さないように配慮されているものが多いようです．また，オブジェクトがROM化可能なものも多いようです．このことは，CP/M-80上でのCプログラムの開発が，移植性よりもアセンブラと比べたソフトウェア開発の効率の向上と組込み型への応用を目的としているためと思われます．また，このような用途にあわせて，多くのコンパイラではライブラリをソースで提供し，ユーザが必要に応じて変更できるようになっています．

16ビット系のCP/M-86上のコンパイラでは，Cの全仕様を満しているものが多いようですが，CPUのアーキテクチャがそれほどC言語に適していないため，コンパイラの能力に，ある程度のばらつきが見受けられます．また，細部では仕様を満していないものもあるようです．

CP/Mでは，OSレベルでI/Oのリダイレクションやパイプ機能はサポートされていません．しかし，いくつかのCコンパイラでは，Cのプログラムが実行を開始する直前に動くスタート・アップ・ルーチン内で引数の処理だけでなく，I/Oのリダイレクションのための〈,〉,〉〉の処理をサポートしています．

この機能があれば,ターミナルとの入出力をファイルとの入出力に実行時に変えることができ，他のOSからCのプログラムを移植する際に，ソースを変更せずに使うことができ便利です．また，パイプ機能に関しては，一時ファイルを経由することにより，擬似的に扱うことはできます．たとえば，

```
A> tr␣'␣'␣'/012'␣<src␣>tmp001
A> sort␣<tmp001␣>tmp002
A> uniq␣<tmp002␣>dst
A> era␣tmp001
A> era␣tmp002
```

とすれば，srcというファイル中のデータをスペースごとに区切ってそれぞれを1行とし(tr)，アルファベット順に並べかえ(sort)，同じ行を取り除き(uniq)，dstファイルに出力します．つまり，srcファイル中の単語をアルファベット順に並べたファイルを作ることが一応できるわけです．

これをUNIXならば，

```
%␣tr␣'␣'␣'\012'␣<src␣|␣sort␣|␣uniq␣>dst
```

の1行で済ますことができます．しかも，パイプの中を通るデータの量が少なければ，これはメモリ上だけで処理されます．この時，tr，sort，uniq の三つのプロセスは並列に動作します．

以上のようにCP/Mでも，Cで記述されたユーザ・プログラムについては，I/Oのリダイレイションやパイプ機能をある程度行わせることができます．

これにシェルらしきものを作れば，一時ファイルもコマンド行から排除することができるわけですが，これはそれほど難かしいことではありません．

ただし，CP/Mでは，階層構造のディレクトリやプロセス管理といったものをサポートするのは困難ですから，CP/Mはあくまで少しはUNIXらしく装うということにとどまります．

MS-DOS

MS-DOSには現在，初期バージョンのV1.25とXENIXからの機能を追加したV2.0があります．V2.0では，UNIXと同じ名前のコマンドがいくつか追加されています．また，標準入出力（I/Oのリダイレクション）とパイプ機能，階層構造のディレクトリが追加されています．これらとV1.25からあるタイム・スタンプ（ファイルごとの変更日時情報）とあわせて，外見はかなりUNIXらしくなってきています．

V2.0では，CP/Mではできなかった，コマンドの出力をパイプにつなぐこともできます．たとえば，

```
A> DIR | SORT | MORE
```

というような組合せも可能です．ただし，V2.0でもMS-DOSはシングルタスクのOSなので，パイプは擬似的に実現されていて，それぞれのコマンドは順番に実行されます．

MS-DOS上で動作するCコンパイラは，CP/M-86上で動作するもののほとんどすべてがMS-DOS用もあるので，その種類はかなり豊富です．

OS-9

OS-9は究極の8ビットCPU，6809専用のOSとして開発されたOSです．レベル1とレベル2があり，レベル2では2Mバイトのメモリ空間を管理できます．

UNIX-likeのOSであり，16ビットのMS-DOSよりもさらにUNIXらしくなっています．具体的にはI/Oリダイレクト，階層構造のディレクトリにくわえて，メモリ内で処理するパイプ機能，マルチタスク，マルチユーザをサポートしており，機能的にはUNIXとほとんど変わりません．

UNIXで，

```
% prog1 < file1 | prog2 | prog3 > file2
```

と書く操作は，

```
OS9 : prog1 < file1 ! prog2 ! prog3 > file2
```

とセパレータ以外は同様に書けますし，並列に動作します．

UNIXの機能を８ビットの小規模なシステムで実現するために，メモリモジュールという概念が取り入れられています．これは，6809ではポジションインディペンデント（位置自由）でリエントラント(再入可能)なプログラムが書けるという特長を生かして，プログラムをすべて特定の形式の位置自由で再入可能なモジュールとして作成することにより，物理メモリ上の１つのモジュールを複数のプロセスで使用するというものです．これによりメモリ効率が良くなり，メモリのスワッピングを行うことなくマルチタスクを実現しています．

OS上で動くプログラムだけでなく，OSそのものも階層構造のモジュールの集合として構成されており，ユーザが必要に応じて自由に追加，変更できます．

モジュールはROM化可能ですから，OSそのものもROM化した小規模の組込みシステムも，ハードディスクを持つ大規模システムも構成できます．

このように裸のOSとしては，現在のパソコンレベルではきわだった高い性能を持つOS-9ですが，トータルな開発環境としてみた場合，UNIXと比較するとかなり貧弱であるのは否めません．UNIXの特徴であるyacc，lex，make等の豊富なツールはなく，Shellの機能も貧弱で，開発環境の中核をなすべきエディタも，UNIXのエディタからは，かなり見劣りするラインエディタです．

しかし，８ビットのフロッピーベースシステムの上である程度UNIXの機能を味わえる点を評価すべきでしょう．

UNIXにあるようなツールは後から付け加えることもできますし，ツールがなくともシングルタスク，平面ディレクトリのCP/M，FLEXなどとは比べられない良好な開発環境を持っているのですから．

FMシリーズ用のOS-9にはマルチウインドウや漢字までサポートされ，BASIC 09という高級言語まで付いています．マイクロウェア社のCは，８ビ

ットのCの中では最高速の部類に属するフルセットのもので，ライブラリ，システムコールも UNIX の C とコンパチビリティが考えられた設計になっています．

各処理系のベンチマークテスト

C言語研究会

最近，各種のパソコンが出回り，そのコンピュータ上でまたさまざまなOSが動作できるような環境ができてきました．そこで問題になってくるのが各OS上の言語の互換性です．ここでは，この互換性が一番高いといわれるC言語のコンパイラについてベンチマークテストを行ってみました．

ベンチマークテストに使用したCと機種

今回のベンチマークテストで使用したOSとCコンパイラは次の通りです．

CP/M	・AZTEC C
	・BDS C COMPILER
OS-9	・MICROWARE C
MS-DOS	・OPTIMIZING C
	・DeSmet C

また，ベンチマークテストに使用した機種構成は次の通りです．

CP/M ……… PC-8801mkII
OS-9 ……… FM-7＋5inch FDD
MS-DOS ……… PC-9801E(8MHz動作)＋8inch FDD

ベンチマークテスト(1)

このテストには次の3つのプログラムを作成し使用しました（**プログラム1，2，3参照**）．このベンチマークテスト用のプログラムを作る際，特に意識したことは，ファイルを取り扱うようなプログラムの場合，各OSのファイル管理の方法やファイルのディスケット上の物理的な位置が，速度測定の際に大きな問題となることです．そのため，今回のベンチマークテストでは，そのようなファイルを取り扱うようなコマンドは使用しないことにしました．

プログラム1は，2重のループだけのプログラムです．このプログラムの目的は，速度はもちろんですが，コンパイルされたプログラムがどのように展開されているかを調べることです．

プログラム1

```
/*                                                */
/*  * It is bench mark test vol.0 *               */
/*                                                */
/*  This sample is loop test                      */
/*                                                */
/*     object size:optimize:time:etc.             */
/*                                                */
/*                  Programed by F-Kaneda         */
/*                                                */

main()

{
    unsigned int i , j ;

    for ( i = 0 ; i < 1000 ; i++ )
        for ( j = 0 ; j < 1000 ; j++ ) ;

}
```

プログラム2

```
/*                                                */
/*  * It is bench mark test vol.1 *               */
/*                                                */
/*  This sample is fanction Ackermann             */
/*                                                */
/*                Programed by F-Kaneda           */
/*                                                */

main()

{
    int x , y , i , ans , ack() ;

    x = 3 ;
    y = 3 ;
    for ( i = 0 ; i < 100 ; i++ )
        ans = ack( x , y ) ;
    printf ( "%d\n" , ans ) ;
}


ack( ax,ay )          /* Function Ackermann */

int ax , ay ;
{
    if ( ax == 0 )
        return ( ay+1 ) ;
    else
        if ( ay == 0 )
            return ( ack( ax-1 , 1 ) ) ;
        else
            return ( ack( ax-1 , ack( ax , ay-1 ) ) ) ;
}
```

プログラム3

```
/*                                                   */
/*  * It is bench mark test vol.2 *                  */
/*                                                   */
/*  This sample is function of include file          */
/*                                                   */
/*              Programed by F-Kaneda                */
/*                                                   */

#include  <stdio.h>
#include  <ctype.h>

main()

{
   char string[256] ;
   char character ;
   int  i , j , k , flag , count ;
   float x , pai = 3.14159 ;

   for ( count = 0 ; count < 1000 ; count++ )
      {
      strcpy( string , "This is test string !!" ) ;
      flag = strcmp( string , "This isn't same string !!" ) ;
      }

   character = '5' ;

   for ( count = 0 ; count < 1000 ; count++ )
      {
      flag = isalpha( character ) ;
      flag = isupper( character ) ;
      flag = islower( character ) ;
      flag = isdigit( character ) ;
      flag = isspace( character ) ;
      }

   for ( count = 0 ; count < 1000 ; count++ )
      {
      x = pai * pai * pai ;
      x = pai / pai / pai ;
      x = pai + pai - pai - pai ;
      }

   for ( count = 0 ; count < 1000 ; count++ )
      {
      j = k = 0x55 ;
      i = ( ( j & k ) == !( !j | !k ) );
      }
}
```

結果は**表1**を見ればわかりますがMS-DOS上のDeSmet Cが一番で次にOPTIMIZING Cが速いことがわかります．ここでCP/M上のAZTEC CやBDS Cがやたら遅いのが気になります．これは使用したPC-8801がDMAやキー割り込みなどにより，実際はZ-80のクロックが4MHz以下になっているのが原因のようです．多分，他のCP/Mではもう少し速い結果が出るでしょう．また，このような短いソースプログラムにもかかわらず，かなりのオブジェクトサイズを食っています．これは，ランタイムルーチンなどをリンクすることによって起こるものです．このソースプログラムなどは各種のチェック（スタックなど）を行う以外は使用しないはずで“コブ”がくっついているようなものです．また，実行速度が異なるOSで大幅に違うのは当然としても，同一のOS上で何割も違ってくるのは不思議なことです．このループのプログラム程度ではオブジェクトの効率がさほど違うとは思えません．やはりこれも何らかのチェックに時間を取られているのでしょう．この差は安全性の代償といえるでしょう．

次に，コンパイラがどのようにソースプログラムを展開しているのか調べてみましょう．まず，AZTEC Cのアセンブルファイルです(**リスト1**)．一見して思うことは「本当にあのプログラムか？」ということです．見事にスパゲッティにミートソースがかかっているといった感じです．かたまりで見ると少しはまともなのですが，この原因は，Z-80のスタック命令が少ないことと，16ビット演算を行うのにHLレジスタを使い，また，ポインタとしてもHLレジスタを使っているからです．これはZ-80の宿命といえるでしょう．それにしてもZ-80はC言語には向いていないようです．OS-9上のMICROWARE Cについても

（35ページにつづく）

表1　プログラム1のベンチマークテストの結果

OS名	コンパイラの名称	オブジェクトサイズ	実行時間
CP/M	AZTEC C	0D80 bytes	126 sec
	BDS C	0880 (0880) bytes	84 (118) sec
OS-9	MICROWARE C	01D0 (01DA) bytes	15 (18) sec
MS-DOS	OPTIMIZING C	4320 bytes	11 sec
	DeSmet C	2048 bytes	9 sec

カッコ内は最適化を行わない場合

リスト1

```
        extrn   .begin,.chl,.swt
        extrn   zsave,zret
        PUBLIC main_
main_:  lxi d,.2
        call zsave
        LXI H,0
        XCHG
        LXI H,6-.2
        DAD SP
        MOV M,E
        INX H
        MOV M,D
        JMP .4
.3:                                 ┐
        LXI H,6-.2                  │
        DAD SP                      │
        PUSH H                      │
        MOV A,M                     │
        INX H                       │
        MOV H,M                     │
        MOV L,A                     │
        INX H                       │
        XCHG                        │
        POP H                       │
        MOV M,E                     │
        INX H                       │
        MOV M,D                     │
.4:                                 │
        LXI H,6-.2                  │
        DAD SP                      │
        MOV E,M                     │
        INX H                       │
        MOV D,M                     │
        LXI H,1000                  │
        CALL .ul                    │
        JZ .5                       │
        LXI H,0                     │
        XCHG                        │
        LXI H,4-.2                  │
        DAD SP                      │
        MOV M,E                     │
        INX H                       │
        MOV M,D                     │
        JMP .7                      │
.6:                     ┐           ├ ループ1
        LXI H,4-.2      │           │
        DAD SP          │           │
        PUSH H          │           │
        MOV A,M         │           │
        INX H           │           │
        MOV H,M         │           │
        MOV L,A         ├ ループ2   │
        INX H           │           │
        XCHG            │           │
        POP H           │           │
        MOV M,E         │           │
        INX H           │           │
        MOV M,D         ┘           ┘
```

```
.7:
        LXI H,4-.2
        DAD SP
        MOV E,M
        INX H
        MOV D,M
        LXI H,1000
        CALL .ul
        JZ .8
        JMP .6
.8:
        JMP .3
.5:
        RET
.2 EQU -4
        extrn   .ul
        END
```


見てみましょう．Cコンパイラの中には，最適化を行うか行わないかを指定できるものもあります．MICROWARE Cもこの1つで最適化をするかしないかを選択できるようになっています．**リスト2**と**リスト3**がそれぞれ最適化する前と最適化後のアセンブラのソースファイルです．最適化後のオブジェクトがかなり短くなっています．よく見るとブランチ命令の直前の命令がブランチ先を変えることによってなくなっているのと，ロングブランチがショートブランチに変わっているのがわかると思います．また，BDS Cなどは，速度を最重視した最適化があり，この場合，速度は速くなるようですがオブジェクトがやや長くなるようです．

リスト2，リスト3は次以降のページに示します．

リスト2

```
                                 psect bench0_c,0,0,0,0,0
0000                             nam   bench0_c
0000                             ttl   main
0000                  main:
0000 3440                        pshs  u
0002 ccffbc                      ldd   #_1
0005=17fff8                      lbsr  _stkcheck
0008 327c                        leas  -4,s
000a 4f                          clra
000b 5f                          clrb
000c ed62                        std   2,s
000e 16001f                      lbra  _4
0011                  _2
0011 4f                          clra
0012 5f                          clrb
0013 ede4                        std   0,s
0015 160007                      lbra  _8
0018                  _6
0018                  _9
0018 ece4                        ldd   0,s
001a c30001                      addd  #1
001d ede4                        std   0,s
001f                  _8
001f ece4                        ldd   0,s
0021 108303e8                    cmpd  #1000
0025 1025ffef                    lblo  _6
0029                  _7
0029                  _5
0029 ec62                        ldd   2,s
002b c30001                      addd  #1
002e ed62                        std   2,s
0030                  _4
0030 ec62                        ldd   2,s
0032 108303e8                    cmpd  #1000
0036 1025ffd7                    lblo  _2
003a                  _3
003a 3264                        leas  4,s
003c 35c0                        puls  u,pc

ffbc                  _1         equ   -68

003e                             endsect
```

リスト2が最適化を行う前，リスト3が最適化後のアセンブラのソースファイルです．よく見比べてください．

リスト3

```
                                  psect bench0_c,0,0,0,0,0
0000                              nam   bench0_c
0000                              ttl   main
0000 3440          main:          pshs  u
0002 ccffbc                       ldd   #_1
0005=17fff8                       lbsr  _stkcheck
0008 327c                         leas  -4,s
000a 4f                           clra
000b 5f                           clrb
000c 2018                         bra   _$2
000e 4f            _2             clra
000f 5f                           clrb
0010 2005                         bra   _$1
0012               _9
0012 ece4          _6             ldd   0,s
0014 c30001                       addd  #1
0017 ede4          _$1            std   0,s
0019 ece4          _8             ldd   0,s
001b 108303e8                     cmpd  #1000
001f 25f1                         blo   _6
0021               _5
0021 ec62          _7             ldd   2,s
0023 c30001                       addd  #1
0026 ed62          _$2            std   2,s
0028 ec62          _4             ldd   2,s
002a 108303e8                     cmpd  #1000
002e 25de                         blo   _2
0030 3264          _3             leas  4,s
0032 35c0                         puls  u,pc
ffbc               _1             equ   -68
0034                              endsect
```

ベンチマークテスト(2)

プログラム2は関数の再帰呼び出しと条件判断のテストです．ここでは有名なアッカーマン関数を定義して，この関数のAck（3，3）を求める動作を100回行っています．答えは61になります．定義式は以下の通りです．

$$\mathrm{Ack}(x, y)=\begin{cases} y+1 & x=0 \\ \mathrm{Ack}(x-1,\ 1) & x>0 \text{ かつ } y=0 \\ \mathrm{Ack}(x-1,\ \mathrm{Ack}(x,\ y-1)) & x>0 \text{ かつ } y>0 \end{cases}$$

結果は，**プログラム1**と同じようですがCP/M-80上で動作する2つのCだけは実行速度が2倍近く上がっています．Z-80は，ループより再帰が得意のようです．また，再帰呼び出しは関数コールの連続と考えることができるため，すべてのプログラムが関数コールであるCの能力を推し量る上で参考になるのではないでしょうか．（**表2**参照）

表2　プログラム2のベンチマークテストの結果

OS名	コンパイラの名称	オブジェクトサイズ	実行時間
CP/M	AZTEC C	1E00 bytes	64 sec
	BDS C	0F00 (0E80) bytes	44 (71) sec
OS-9	MICROWARE C	0EC3 (0EE2) bytes	14 (15) sec
MS-DOS	OPTIMIZING C	4448 bytes	8 sec
	DeSmet C	2048 bytes	7 sec

カッコ内は最適化を行わない場合

ベンチマークテスト(3)

プログラム3はCコンパイラ内部の組み込み関数と実数演算，論理演算などについてのベンチマークテストです．本当は，三角関数や対数などもやってみたかったのですが，これらの関数を持っていないものが多いために今回は遠慮しました．また，BDS Cはコンパイラ本体だけでは実数演算ができず，外部の実数計算用のヘッダをインクルードしなければならないために，ソースに互換性がなくなるため，このテストでは避けました．

結果は，前のテストとほとんど同じですが，DeSmet Cだけが速くなっています．このコンパイラは実数の演算パッケージなどが速くなるように作ってあるのでしょう．(**表3**参照)

表3　プログラム3のベンチマークテスト

OS名	コンパイラの名称	オブジェクトサイズ	実行時間
CP/M	AZTEC C	2200 bytes	104 sec
	BDS C	—— bytes	—— sec
OS-9	MICROWARE C	0BBD (0BDB) bytes	21 (22) sec
MS-DOS	OPTIMIZING C	6681 bytes	14 sec
	DeSmet C	4608 bytes	5 sec

カッコ内は最適化を行わない場合

コンパイル速度のテスト

すべてのテストに対して，OS-9のMICROWARE Cは15秒前後です．また，オブジェクトもかなり小さくなっているのが特徴的です．MS-DOS上の2つのCにおいては，たいへんオブジェクトが長くなっているようですが，これは多機能なランタイムルーチンをリンクしているため，このようになってしまうのでしょう．メモリを多くとれるOSはサイズを気にしなくていいため，開発なども楽になっています．

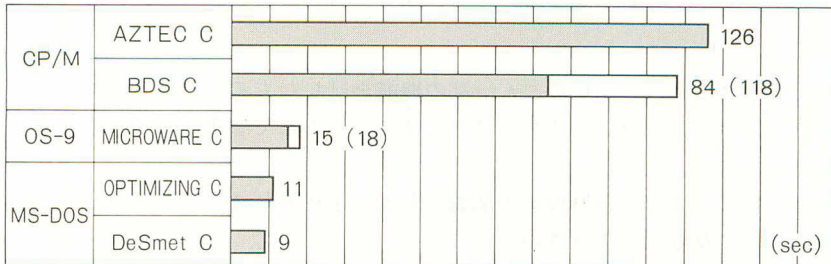
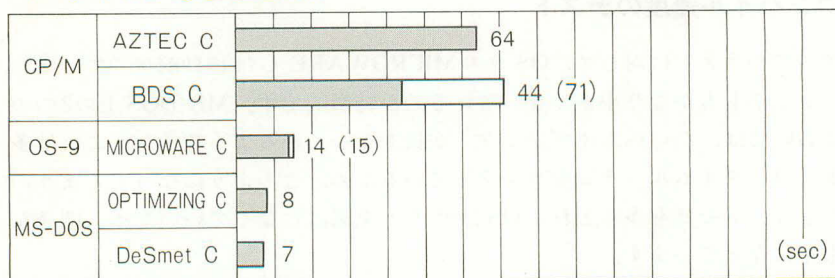
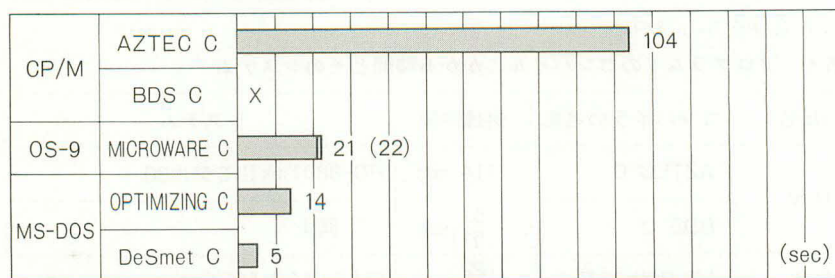
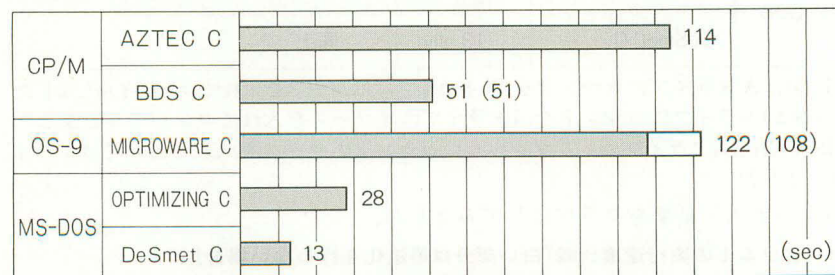
各Cコンパイラのコンパイル速度はそのファイル構成や使用する周辺機器によって異なってくるものですが，1つの例として**表4**にコンパイル速度とそのテスト条件を示します．

表4　プログラム1のコンパイルにかかる時間とそのシステム

OS名	コンパイラの名称	処理時間	システム
CP/M	AZTEC C	114 sec	PC-8801 mk II モデル30
	BDS C	51 (51) sec	同上
OS-9	MICROWARE C	122 (108) sec	FM-7＋LFD 550(EXA)ステップレート6ms
MS-DOS	OPTIMIZING C	28 sec	PC-9801E＋PC-9881K(8インチ)
	DeSmet C	13 sec	同上

・CP/MはAドライブにコマンドを，Bドライブにはソースを入れてコンパイルしました
・OS-9は0ドライブにコマンドを，1ドライブにはソースを入れてコンパイルしました
・MS-DOSはAドライブのトップレベルにコマンドとソースを入れておいてコンパイルしました

以上のテスト結果をグラフにしてみました．

プログラム1の実行速度比較(白い部分は最適化を行わない場合)

プログラム2の実行速度比較

プログラム3の実行速度比較

プログラム1のコンパイルにかかる時間の比較

ここにあげたテストはCのごく一部の機能を試したものに過ぎません．その意味で，このベンチマークテストはあくまで一例として参考にしていただきたいと思います．

本書の巻末には，「各処理系の関数一覧表」を掲載しました．各社Cの特徴を知るうえで参考にして下さい．

各処理系の仕様一覧

C言語研究会

BDS C COMPILER

CPU	8080, 8085, Z-80
OS	CP/M-80
開発元	BD SOFTWARE INC.
代理店	(株)ライフボート　　TEL(03)456-4101
定価	¥60,000—

特徴

CP/M-80用としては一番普及しているCコンパイラ．floatをサポートしていないサブセット仕様ですが，高速コンパイルおよび高速オブジェクト等の利点を生かしてアセンブラの代替としてよく使用されます．

強力なユーザーズグループに支えられているのも強味．ちなみにBDはBrain Damageの略です．

AZTEC CII

CPU	Z-80, 8080
OS	CP/M-80
開発元	MANX SOFTWARE SYSTEMS
代理店	(株)サザンパシフィック　　TEL(045)501-8842・8919
定価	￥47,500—

特徴

カーニハン＆リッチーの仕様をフルサポートするコンパイラ．BDS C とは対照的に機能の豊富さを目指しています．しかしコンパイルおよびオブジェクトはBDSに比べると遅いようです．

上位機種のソースをZ-80用に落とすのに向いています．

OPTIMIZING C86 C コンパイラ

CPU	8086
OS	MS-DOS ver.1.25,　ver.2.0,　CP/M-86
開発元	Computer Innovations Inc.
代理店	マイクロソフトウェア　　TEL(03)813-8221(代)
定価	――――――

特徴

8086用として大変魅力のあるCコンパイラです．すべてのソースファイルが標準で付いてきます．また日本語処理機能，グラフィック機能も有しているうえ，UNIX Cとのコンパチビリティも他に勝っています．半面，コンパイル速度，オブジェクトサイズはやや劣ります．

MS-DOS ver.2.0 以後の階層ディレクトリ構造などをサポートしています．アーカイブ・ユーティリティが付いてます．

DeSmet C

CPU	8086，8088
OS	CP/M-86，MS-DOS ver.1.25，ver.2.0
開発元	DeSmet Software
代理店	ソフトウェア・インターナショナル(株)　　TEL(03)486-7151
定価	¥92,000―

特徴

8086用としては安価な部類に入るコンパイラ．代理店によってエンハンスされており漢字使用可能になっています．他に1日無料セミナー，電話によるコンサルタント，日本語マニュアル等々，代理店の力の入れ方が目につきます．機能は全体にやや小ぶりな印象を受けます．専用のフルスクリーンエディタが附属として付いています．

MICROWARE C

CPU	6809
OS	OS-9 Level I, II
開発元	MICROWARE INC.
代理店	マイクロウェアジャパン株式会社　TEL0473(28)4493
定価	¥160,000—

特徴

OS-9 用唯一のフルセットCコンパイラです．8bit 用でマルチタスクおよびマルチユーザ機能を唯一サポートしているCコンパイラです．逆に算述関数がほとんどないのは残念です．使い勝手がよい点としてヘッダファイルのディレクトリが固定されていること，コンパイラが自分で procedure file を作りそれを実行すること，UNIX の cc のようにCソース，アセンブラソース，オブジェクトの各ファイルを任意の数並べて引き渡せることなどがあります．

第2部

Cトレーニングサンプル編

本編の読み方

第1部では，パソコン上でC言語を走らせた場合の作業環境や各OS，CPUとの相性について解説したほか，各処理系のベンチマークテストのデータを公開しましたが，本編では，実際にパソコン上で走るサンプルプログラムを紹介します．

これらのサンプルプログラムは，実用を目的としたものではなく，あくまで初歩的なプログラミングテクニックの習得を目的として書かれたものです．そのため，プログラムとしての完璧さよりも，主要な関数がいかにプログラム上で使われているかに重点を置いて説明する形をとりました．

プログラミングのテクニックを習得するにあたって，まずC言語の典型的なプログラムのスタイルや基本的な関数の機能を知る必要がありますが，本編の構成は，数ある関数の中から最も基本的，しかも頻繁に使われる関数から順を追って説明するよう配慮しました．基本的な関数とは同時に最も簡単な関数でもあります．単純なものから複雑なものへと，ステップを踏みながら進めていくようになっています．

本編では，プログラムのスタイルおよび関数の使われ方の解説は，次のような順で流れていきます．

BASICのprint文に相当するprintf文およびinput文に相当するgetchar文を使ったもの．鶴亀算を例にとった四則演算では，ただ単に動作するプログラムだけにはとどまらず，プログラムを整然と見やすくする方法やプリプロセッサを用いた場合などを示しました．条件判断のif～else文の応用や論理演算子といった特殊な演算子の考え方の解説．同じプログラムを多重分岐else if文とswitch文で作った場合．ループを使い，各種のプログラムに応用したものなどを順を追って解説していきます．

メッセージ　　関数printfを使う

```
/* PRINT MESSAGE */

main()
{
     printf("Welcome to C programing!\n");
}
```

このプログラムは文字列を印字するためのものです．このプログラムを実行すると，ディスプレイ上に””(ダブルクォート）で囲んだ文字が出力されます．

Cでは，プログラムはすべて関数から出来ています．つまりmain( )というのは「メインプログラムを実行する」という働きを持つ1つの関数です．一般に関数といえば，Y＝2X＋3のようにある値（引数）に対して，対応する値を返してくるものですが，このmain( )というのはカッコの中に何もありませんから，引数のない関数ということになります．そのすぐ下と一番最後にある{}（中カッコ）は1つの関数がどこからどこまでかを示すものです．；（セミコロン）は1文がここで終わることを表しています．

通常，メインプログラムは他の関数を順次呼び出すことによって実行されます．その関数は，プログラマが自分で書いたり，Cに標準装備されているライブラリ関数を使用したりします．ここでは標準入出力関数であるprintfを使っています．ここでのprintfはダブルクォートで囲まれた文字列を引数として，それを出力する関数となっています．

文字列の最後にある＼n（バックスラッシュエヌと読む）は，改行を行うための文字です．printfでは改行が自動的には行われないため，＼nをつけないと，出力した文字は次々につながってしまいます．なお，JISのキーボードには＼がありませんので￥でも代用できます．この＼nのようなエスケープ文字は他に＼b（バックスペース），＼t（タブ），＼”(ダブルクォート），＼’(シングルクォート），＼＼(バックスラッシュ），＼0（ヌルキャラクタ）等があります．各々試してみるとよいでしょう．

メインプログラムの頭にある／＊　PRINT　MESSAGE　＊／　はコメント文です．／＊から＊／までの文字はプログラムでは無視されますので，プログラムをわかりやすくするためになるべく利用するようにするとよいでしょう．

鶴亀算(1)　Cにおける四則演算

```
/* TSURUKAMEZAN */

main()
{
      short crane, turtle;

      crane = (300 * 4 - 840) / 2;
      turtle = 300 - crane;
      printf("TSURU %3d KAME %3d\n", crane, turtle);
}
```

「鶴と亀が合わせて300頭おり，その足の数の合計は840本です．さて鶴と亀はそれぞれ何頭いるでしょうか」．子供の頃こんな問題を出されたことはありませんか．この問題を解く公式は，

〈鶴の数〉＝(〈総数〉＊4－〈足の総数〉)／2
〈亀の数〉＝〈総数〉－〈鶴の数〉

で与えられます．この式を使って作ったものが上のプログラムです．実行すると“TSURU＝180　KAME＝120”と解答が表示されます．

ここでは鶴の数を表すcrane，亀の数を表すturtleという2つの変数が使われています．Cでは使用する変数のすべてはその関数の始めのところで宣言をしておく必要があります．short crane，turtle；とあるのがその宣言文です．

宣言文ではデータの型と，その型を持つデータが，(コンマ)で区切られて並べられます．ここでいうデータとは，変数，関数，配列なども含まれます．データ型には，

char	(文字型)
int	(整数型)
float	(浮動小数点型)
double	(倍精度浮動小数点型)

の4つがあり，このうち整数型にはintの他に，

short int	(短整数型)
long int	(長整数型)
unsighned int	(符号なし整数型)

があります．これら３つの型ではintの文字は省略可能で，省略するのがほとんどです．整数型では，機械によってはintがshortまたはlongと全く同義である場合があります．これはintがその機械が普通に処理する整数の長さによって決まるためです．自分の使っているコンパイラのマニュアルをよく読んで，どの型がどれだけの長さ（ビット）を持っているか，よく調べてください．

Cでの四則演算は，ほかの言語と変わりなく＋，－，＊，／で＊，／が優先されます．カッコの使い方も同様です．なお，整数の割り算では小数部分は切り捨てられます．これらを使って鶴，亀の数を計算しています．

このプログラムのprintfでは文字列だけでなく変数の値を出力しています．変数または式の値は上の例のように％で始まる変換指示子と呼ばれるものによって，どの位置にどのような形式で出力されるかを決定されます．上の例では，始めの％3dは１番目のturtleの値を整数3桁で出力することを示しています．同様に％5.3fは浮動小数点型の値を5文字幅で小数点以下３桁まで出力することを表しています．％に続く文字は他に，o（８進数），x（16進数），c（文字），s（文字列），％（'％'を出力）等があります．

鶴亀算(2)

鶴亀算(1)のプログラムをもっと見やすく

```
/* TSURUKAMEZAN ver.2 */

main()
{
     short sum, leg, crane, turtle;

     sum = 300;
     leg = 840;

     crane = (sum * 4 - leg) / 2;
     turtle = 300 - crane;
     printf("TSURU %3d KAME %3d\n", crane, turtle);
}
```

鶴亀算(1)のプログラムを少し変えたものです．実行結果はまったく同じです．前のプログラムでは，鶴と亀の総数や足の本数が実際にどこに代入されているのかはすぐにはわかりません．こうすればプログラムが見やすく，他の人にもわかりやすくなります．このように，定数であっても，その定数がある特定の意味を持つ場合には，変数名をつけてメインプログラムの始めのところで値を初期値として代入文を書いておく習慣をつけておくとよいでしょう．

鶴亀算(3)　プリプロセッサを使う方法

```
/* TSURUKAMEZAN ver.3 */

#define    SUM  300
#define    LEG  840

main()
{
     short crane, turtle;

     crane = (SUM * 4 - LEG) / 2;
     turtle = 300 - crane;
     printf("TSURU %3d KAME %3d\n", crane, turtle);
}
```

これも前のプログラムと同様に変更を加えたものです．前のプログラムでは定数を変数として前に出しましたが，これはプログラムの先頭に出してしまう方法です．＃defineというのはマクロ定義を行うもので，いうならば文字列を置き換えるものです．上の例ではSUMとLEGがそれぞれ300，840に置き換えられています．ですからプログラム中でわざわざ変数を宣言する必要がありません．マクロ定義した文字は区別するために大文字で書くのが普通で，その有効範囲は，定義が行われたところからプログラムの最後まででです．途中で定義し直すことも可能です．ただ” ”で囲まれた文字列では置き換えが行われません．試しに上のプログラムに，printf(”SUM/n”)；という文を挿入しても300とは出力されません．各自で試みてください．

実は，＃define文はプリプロセッサと呼ばれるものの一種です．プリプロセッサというのはコンパイルが始まる前に参照される指定のようなものと考えると理解しやすいでしょう．プリプロセッサは＃defineのほかに，＃include，＃if，＃ifdef，＃ifndef，＃else，＃endif，＃lineがあります．

＃include文はあるファイルをファイル名をつけて指定し，そのファイルの内容全体と＃include文を置き換える働きをします．つまり，プログラムの最初または途中に＃include ”test.h”という文があると，その箇所にtest.hのファイルの内容がそのまま挿入されたのとまったく同じことになります．これはよく使われる＃define文の集合などをファイルにしておいて，コンパイルするときにプログラムの最初に呼び出して使う，という使われ方が多いようです．

＃if，＃ifdef，＃ifndef，＃else，＃endifは「条件付きコンパイル」を行うときに利用されます．また＃lineは仮の行番号とファイル名を与えるものです．

ここは少しわかりづらいので説明は省略します．もう少しCの構文を覚えてから勉強した方がよいでしょう．

プリプロセッサは，このように普通のプログラムとはちょっと違った働きをします．それはプリプロセッサの後にセミコロンがないことからもわかると思います．

2文字の入れ替え　標準入出力関数

```
/* REVERSE 2 CHARACTERS */

main()
{
      short a, b;

      a = getchar();
      b = getchar();
      putchar(b);
      putchar(a);
}
```

これは非常に単純な，入力した2文字を入れ替えて出力するプログラムです．ここで使われているgetchar，putcharの2つの関数はそれぞれ端末に対し入出力を行う関数です．入出力を行う対象はファイルにすることも可能です．

getchar(　)は入力された文字の初めの1文字を持ってくる関数なので，これをいくつか並べれば，それだけの文字を取り込むことができます．putchar(　)はちょうどその逆の働きをしますが，こちらは何の値を出力するのかを引数として渡す必要があります．

プログラムを実行させて何文字か入力してリターンしてみてください．2文字より多く入力しても後了の文字は無視されてしまいます．逆に1文字しか入力しない場合は2文字目は改行コードが入り，1行あけて1文字だけを出力します．これはリターンキーを押したことがgetchar(　)で読み取られるためです．

実行例

①A B↵	②A↵	③A B C D↵	………(入力)
B A		B A	……(出力)
	A		

数字のチェック(1)　if〜else文

```
/* CHECK CHARACTER WITH NUMBER */

main()
{
     short a;

     a = getchar();
     if (a >= '0') {
          if (a <='9')
               printf("It's numeral.\n");
          else
               printf("It's not numeral.\n");
     }
     else
          printf("It's not numeral.\n");
}
```

入力した文字が数字(0〜9)であるかどうかを調べて，数字であれば "It's numeral." そうでなければ "It's not numeral." とメッセージを出力するプログラムです．

このプログラムでは数字かどうかを判定するために，if〜else文を使っています．この構文は，

```
if （式）
     プログラム1
else
     プログラム2
```

で表わされ，式の値が真(0でない)のときはプログラム1を，偽(0)であるときはプログラム2を実行します．else部は省略可能です．各々のプログラムは1文でも，中カッコ付きの複数の文でもかまいません．if〜else文を書く場合，それに続く文は1段落として書き，どのifがどのelseに対応しているかをはっきりさせて書くようにします．

(式)にはどんな式を書いてもかまいませんが，よく使われるのはやはり関係演算子です．>（大きい），>=（大きいか等しい），<（小さい），<=(小さいか等しい)，==（等しい），!=（等しくない）があり，このうち==と!=は他の4つに比べて優先度が1段低いため，いっしょに扱う場合には注意が必要です(ちょうど+－*/の関係と同じです)．

上のサンプルプログラムでは，始めのif文でaと'0'の大きさを比べています．Cでは文字定数は' '(シングルクォート)で囲まれた1文字で表わされ，その文字のコードを値として持ちます．よってaと'0'の大小とはそれらの文字コードの値の大小を表しているのです．数字のコードは'0', '1', ………'9'と続いていますから，aが'0'より小さければそれは数字ではなく，1番下のelse文に飛んで"It's not numeral."と表示されます．aが'0'以上の場合は次のif文に移り，今度は'9'と比べます．aが'9'より大きい場合は，やはり数字ではありませんので，対応するelse文に飛んで"It's not numeral."と出力します．aが'9'以下の場合は数字であることがわかりますから，ここで始めて，"It's numeral."と表示されるわけです．

数字のチェック(2)

論理演算子

```
/* CHECK CHARACTER WITH NUMBER ver.2 */

main()
{
    short a;

    a = getchar();
    if (a >='0' && a<='9')
        printf("It's numeral.\n");
    else
        printf("It's not numeral.\n");
}
```

前のプログラムは同じ文が2つあるなど結構むだがありました．それを簡潔にまとめたのが上のプログラムです．動作はまったく同じです．

ここでは論理演算子が使われています．if文の式を見てください．&&というのは論理積(かつ)を表します．つまり「aが'0'以上でかつaが'9'以下」のとき"It's numeral."が表示され，それ以外のときは"It's not numeral."が表示されます．これは前のプログラムと論理的に同じことをいっているのです．

このような論理演算子は他に||（論理和：または）があり，&&や||で連続した式は左から右へ評価されます．

なお，if文では条件式の後に；（セミコロン）は付けません．付けてしまうと条件が真となっても実行する文がなくなってしまいます．注意してください．

1日のあいさつ(1)　else if文

```
/* PASS THE TIME OF DAY */

main()
{
        short a;

        a = getchar();
        if (a == 'm' || a == 'M')
                printf("Good morning.\n");
        else if (a == 'a' || a == 'A')
                printf("Good afternoon.\n");
        else if (a == 'e' || a == 'E')
                printf("Good evening.\n");
        else
                printf("Good night.\n");
}
```

対応する英文字1字を入力することによって，"Good morning.", "Good afternoon.", "Good evening.", "Good night." とコンピュータがあいさつを返すプログラムです．

ここでは多重分岐を行うためにelse if文を使っています．これは前に勉強したif～else文のelse節にもう1つif文を続けた形になっています．

```
if （式1）
    プログラム1
else  if （式2）
    プログラム2
        ⋮
else
    プログラムn
```

たとえば，上のサンプルプログラムを実行して'A'を入力すると，その値が変数aに代入され，初めのif文の式が評価されます．aは'm'でも'M'でもありませんから，else節にスキップし，次に続くif文の式が評価され，a＝＝'A'の式が真となりますから，次の文を実行（"Good afternoon." を出力）して終わります．

このようにelse if文では，条件式が真になるものを捜して次々とスキップしていき，すべての条件に合わなかった場合は最後のelse節の文を実行する働きを持っています．

1日のあいさつ(2)　switch文

```
/* PASS THE TIME OF DAY ver.2 */

main()
{
	short a;

	a = getchar();
	switch(a) {
	case 'm':
	case 'M':
		printf("Good morning.\n");
		break;
	case 'a':
	case 'A':
		printf("Good afternoon.\n");
		break;
	case 'e':
	case 'E':
		printf("Good evening.\n");
		break;
	default:
		printf("Good night.\n");
		break;
	}
}
```

前のプログラムをswitch文を使って書き直すと上のようになります．

これと前のプログラムを見比べてもらえばよくわかると思いますが，switch文は，まず直後のカッコの中の式を計算し，その値がどのcaseに当てはまるのかをすべて調べます．当てはまるcaseがあればそのcaseで文が実行され，すべてのcaseに当てはまらない値の場合はdefault：の文が実行されます．case, defaultの並ぶ順はどちらが先でも結構です．defaultの部分は省略可能です．

breakとあるのはswitch文から強制的に抜け出す文です．多重分岐でなぜこれが必要かというと，switch文ではある当てはまるcaseが存在した場合，そこから下の文はすべて実行してしまうからです．上の例でいえば，もしすべてのbreak文がない場合，'M'を入力すると「おはよう」から「おやすみ」まで一気にあいさつされて面喰うことになります．これは特殊な使い方では有効かも知れません．あまり使わないほうがよいかもしれません．

またdefaultがないときは最後のcaseの後にもbreak文をつけるようにするとよいでしょう．これは何の意味もないように思えますが，将来caseを加えるときのちょっとしたバグ防止法になるからです．

整数の和を求める(1)　while文, インクリメント演算子

```
/* SUM OF NUMBER 1-50 */

main()
{
     short sum, i, limit;

     sum = i = 0;
     limit = 50;

     while (i <= limit) {
          sum = sum + i++;
     }
     printf("SUM = %4d\n", sum);
}
```

1～50（プログラムでは0から）の整数の和を求めるプログラムです．

このプログラムではループを行うためにwhile文を使っています．while文の形式は，

```
while （式） {
    プログラム１
}
```

で与えられます．プログラム１が１文のときは中カッコは省略可能です．whileではまず式が計算され，その値が真（０でない）のときプログラム１を実行し，再度式を計算してその値が偽(０)になった時点でループを抜けます．

サンプルプログラムを見るとまず変数の初期化が行われています．ここで目新らしいのはsum＝i＝0;という表現です．これは代入文が値を持ち，左から右へ代入されるためにこういうことが可能となるのです．つまり，まずi＝０でiに０が代入され，またi＝０という文が０という値になるのでsumの値も０になるのです．これは多くの変数に同じ値を一度に代入できるので便利です．

次にwhile文の中にi＋＋という表現があります．これはインクリメント演算子と呼ばれ，変数の値を１増やすものです．＋＋の記号は変数の前にも後ろにも付けることができますが，sum＋＋＋iはiを１増やしてからsumを加えることを表すのに対して，sum＋i＋＋ではiをsumに加えてからiを１増やす，というように，インクリメントが行われる時期が違ってきます．

同じ用法で１を引く働きを持つデクリメント演算子（－－）もあります．

整数の和を求める(2)　for文，代入演算子

```
/* SUM OF NUMBER 1-50 ver.2 */

main()
{
     short sum, i, limit;

     sum = 0;
     limit = 50;

     for (i = 0; i <= limit; i++)
          sum += i;
     printf("SUM = %4d\n", sum);
}
```

前のページのプログラムをfor文を使って書き換えてみました．他にも少し変えたところがあります．

for文とwhile文は書き方が違うだけで働きはほとんどいっしょです．どちらを使うかは好みの問題といえるでしょう．for文の形式は，

```
for( 1 式 ; 2 式 ; 3 式)
     プログラム 1
```

となっています．1式は初期設定の式で，上の例ではi＝0となっています．2式は実行条件で，この式の値が真(0でない)のときはプログラム1が実行されます．そして次に3式の再初期化が実行されてまた2式が評価されます．つまりwhile文で書くと，次のようになります．

```
1 式 ;
while  ( 2 式)  {
     プログラム 1
     3 式 ;
}
```

また，for文の中の3つの式はどれも省略可能ですが;(セミコロン)は省略することはできません．

for文の次にあるsum＋＝i;というのは，代入演算子と呼ばれるもので，sum＝sum＋i;とまったく同じことです．これは「sumに加えるのはｉ」と考えるとわかりやすいでしょう．この他に－＝，＊＝，/＝等が使えます．

整数の和を求める(3)　do～while文

```
/* SUM OF NUMBER 1-50 ver.3 */

main()
{
	short sum, i, limit;

	sum = i = 0;
	limit = 50;

	do {
		sum += i++;
	} while (i <= limit);
	printf("SUM = %4d\n", sum);
}
```

ループを作るにはもう1つdo～while文があります．これを使って整数の和を求めるプログラムを書くと上のようになります．

do～while文の形式は，

```
do
    プログラム1
while （式）；
```

となります．式についてはwhile文のときと全く同じです．

do～while文がwhile文やfor文と違う点は，whileやforがループの初めのところで条件判断を行うのに対し，do～while文ではループの一番最後で条件判断を行う点です．ですから，どんな条件を与えても1回はループの中の文を実行することになります．試しに今までの3つのプログラムのlimitの値を0，iの初期値を1にしてコンパイルしてみてください．前の2つのプログラムでは，条件判断の値が偽(0)となってしまうために，sumにiの値を加えずに結果を出力し，結果は0となります．しかしdo～while 文を使ったプログラムでは，sumにiの値を加えたあと条件判断を行いますから，結果は1となります．

このように，Cには3種類のループがあります．各々の特徴をつかんで見やすく効率のよいプログラムを組んでください．

幾何模様　for文のネスティング

```
/* GEOMETRICAL PATTERN No.1 */

main()
{
	short i, j;

	for (i = 1; i <= 10; i++) {
		for (j = 1; j <= i; j++)
			printf("$");
		printf("\n");
	}
}
```

実行例

```
$
$$
$$$
$$$$
$$$$$
$$$$$$
$$$$$$$
$$$$$$$$
$$$$$$$$$
$$$$$$$$$$
```

```
/* GEOMETRICAL PATTERN No.2 */

#define   PI  3.1415926

main()
{
	short i, j, step, width;
	double k, sin();

	step = 10;
	width = 20;

	for (i = 0; i <= 90; i += step) {
		k = sin(i * PI / 180.0) * width;
		for (j = 1; j <= k; j++)
			printf("$");
		printf("\n");
	}
}
```

実行例

```
$$$
$$$$$$
$$$$$$$$$$
$$$$$$$$$$$$
$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$
```

上のプログラムは実行例のような模様を描くプログラムです．for文2つを入れ子（ネスティング）させて，iの値によって'$'を並べる数を決定しているわけです．下のプログラムでは値の変化のために三角関数を使ってみました．

三角関数，その引数は共に倍精度浮動小数点の値を持つため，初めのところで宣言をしておきます．三角関数の引数の単位はラジアンですからカッコの中に変換する式が書いてあります．このカッコの中では整数型と浮動小数点型がいっしょになっていますが，こういうときは，整数型は自動的に浮動小数点型に変換されるために，めんどうなことは何も起こりません．

奇数の和を求める　continue文

```
/* SUM OF ODD NUMBER 1-100 */

main()
{
     short sum, i, limit;

     sum = 0;
     limit = 100;

     for (i = 1; i <= limit; i++) {
          if (! (i % 2))
               continue;
          else
               sum += i;
     }
     printf("SUM = %4d\n", sum);



/* SUM OF ODD NUMBER ver.2 */

main()
{
     short sum, i, limit;

     sum = 0;
     limit = 100;

     for (i = 1; i <= limit; i += 2)
          sum += i;
     printf("SUM = %4d\n", sum);
}
```

1から100までの数のうち，奇数のみを加えていくプログラムを2種類作ってみました．プログラムとしてはもちろん下のプログラムの方がまとまっています（もっとまとめることもできます）が，上のプログラムには新しい構文がいくつか含まれていますので，その説明のために出してみました．

！(式)というのは論理否定演算子のことで，式の値が真（0以外）のときは0の値となり，偽(0)のときは1となります．つまりif(！(i%2))は，if((i%2)＝＝0)と同義です．

%はモジュロ演算子と呼ばれるもので，左の値を右の値で割った余りを値として持ちます．

continue文はループの文をそこでストップさせ，ループの頭に戻る動作をさ

せる文です．forループであれば，continue文が来ればすぐ再初期化の式を実行して条件判断を行います．

ですから上の例では，ｉを２で割った余りが１のときはsumにｉを加え，０のときはなにもしないでループの再初期化（ｉのインクリメント）が行われ，これを続けていき奇数の和を求めているのです．

整数の和と奇数の和　for文の特殊な使い方

```
/* SUM OF NUMBER 1-50 AND ODD NUMBER 1-100 */

main()
{
	short sum1, sum2, i, j, limit;

	sum1 = sum2 = 0;
	limit = 50;

	for (i = 1, j = 1; i <= limit; i++, j += 2) {
		sum1 += i;
		sum2 += j;
	}
	printf("SUM1 = %4d  SUM2 = %4d\n", sum1, sum2);
}
```

上のプログラムは，前述の整数の和と奇数の和を１つのfor文で求めようという，小々乱暴なプログラムです．このプログラムももっとわかりやすく書けるのですが，ここではfor文の特殊な使い方の例として見てください．

Ｃでは，（コンマ）で連結された式の並びは左から計算され，一番右の式の値を持ちます．これをコンマ演算子と呼びます．そしてこれをfor文に利用すると並列処理を行うことができるのです．

プログラムを見てわかるように，ｉとｊの２個のパラメータを，ｉは１ずつｊは２ずつ別個に増やして，それぞれの合計に加えています．コンマ演算子の性質から，初期化の箇所の式と再初期化の式のどちらか，または両方ともｉとｊの式を入れ替えても，動作が同じであることが理解できると思います．

また同じコンマでも，宣言文での変数等を分けたりするコンマは演算子ではありませんので注意してください．

円の面積

scanf関数とポインタ

```
/* AREA OF CIRCLE */

#define    PI    3.1415926

main()
{
        float radius;

        do {
                scanf("%f", &radius);
                printf("AREA = %10.3f\n", PI * radius * radius);
        } while (radius != 0.0);

}
```

円の面積を求めるプログラムです．半径の値を入力すると，その半径の円の面積が出力されます．0を入力すると終了します．

ここではscanfという関数が登場します．scanfはちょうどprintfの逆の働きをするもので，変数に数値あるいは文字列を入力する関数となっています．"%f"というのは，数値を浮動小数点として変数に取り組みなさいということです．scanfもprintfと同様に%に続く文字としてd（10進数），o（8進数），x（16進数），c（文字），s（文字列）等が使えます．

しかし，scanfのもう1つの引数を見ると，&radiusと変数名の前に&がついています．&は次に続く変数のアドレスを示す単項演算子です．つまりscanfでは変数の値を引数とするのではなく，その変数のアドレスを引数とするのです．

ここで少し，ポインタというものについて説明をしましょう．ポインタというのはあるデータ型の変数（または配列）のアドレスを持っている変数です．ポインタは通常その先頭に＊が付いた形式で型宣言されます．たとえば，

```
flort   var, *varp ;
```

という宣言をすると，varp＝&var;という式で浮動小数点型の変数varのアドレスがvarpに代入されます．また＊varpはvarpが指し示している内容，つまりvarそのものと何ら変わりないものとなります．＊varp＝13. 5;と書けば，varに13.5という値が代入されます．

ポインタについては，これからも出てきますので，少し頭の中にとどめておいてください．

入力値の割合

配列の使い方

```
/* RATE OF INPUT NUMBER */

main()
{
	short i, j, sum;
	short a[256];

	sum = 0;

	for (i = 0; i < 256; i++) {
		scanf("%d", &a[i]);
		if (a[i] == 0)
			break;
		else
			sum += a[i];
	}
	for (j = 0; j < i; j++)
		printf("%5d  %5.2f%%\n",
			a[j], a[j] * 1.0 / sum * 100.0);
}
```

整数の値を0～32767の範囲で入力して，最後に0を入力すると，入力した数値と，全体に対する割合を表示するプログラムです．入力できる数値の数は256個までです．

このプログラムでは配列を使っています．配列の宣言は，変数と同様に型宣言を行い，配列名のあとに大カッコで囲まれた，配列の要素の個数が示されます．Cでは配列の添字は0から始まるため，上記のプログラムではａ［0］～ａ［255］までの256個の短整数型の配列が宣言されたことになります．

Cにおける配列名はそのまま0番の要素のアドレスを示します．これはポインタと似ていますが，ポインタが変数であるのに対して，配列名は変数ではない点が違います．またａ＋ｉはａの配列のｉ番の要素のアドレスを示すので，上のプログラムの中のscanfの引数である&a[ｉ]はそのままａ＋ｉに置き換えることが可能です．同様にａ[ｉ]は＊(ａ＋ｉ)と置き換えることができます．

プログラムの最後の計算では整数型の配列を浮動小数点型に変換するためにわざわざ1.0を掛けています．こうしないと，整数型同士で割り算を行う場合，小数点以下が切り捨てられてしまうからです．

階乗計算(1)　新しい関数を作る

```
/* FACT */

main()
{
      short n;
      long ffact();

      printf("INPUT NUMBER  ");
      scanf("%d", &n);
      printf("ANSWER = %ld\n", ffact(n));
}
long ffact(n1)
short n1;
{
      short i;
      long f = 1;

      if (! n1)
            return(1);
      else
            for (i = 1; i <= n1; i++)
                  f *= i;
            return(f);
}
```

これは階乗を求めるプログラムです．数を入力すると"ANSWER＝"と答えが表示されます．

ここでは独自に階乗を求める関数ffactを作っています．この関数はlong型の値を返してきますから，関数名の箇所と，メインプログラムで型の宣言が必要になります．通常の整数型の場合，これらの宣言は省略可能です．

関数名の次に，引数の型の宣言が必要です．これは中カッコの前に行います．後はmain(　)と同じように，引数の他に使用する変数を宣言して，文を書けばよいのです．

return文はカッコの中の値を関数の値として返す文です．ですからffactを呼んできた場合は，その引数が0のときは1の値を返し，そうでないときはfor文によって階乗を計算し，その結果の値を返します．

Cでの引数は値が与えられるだけなので，引数の値を関数内で変えてしまってもメインプログラムでの値はまったく変わりません．これをcall by value（値による呼出し）と呼んでいます．

階乗計算(2)　再帰呼び出し，条件演算子

```
/* FACT ver.2 */

main()
{
      short n;
      long ffact();

      printf("INPUT NUMBER  ");
      scanf("%d", &n);
      printf("ANSWER = %ld\n", ffact(n));
}
long ffact(n1)
short n1;
{
      return(n1 == 0? 1: n1 * ffact(n1 - 1));
}
```

階乗を求める関数を再帰呼び出しを使って書き換えると，上のプログラムとなりました．

再帰呼び出しとは，関数が値を計算する際に自分自身を他の関数のように呼び出して使うことです．ffactの関数の中でffactが呼ばれているのがわかると思います．

また，ここでは条件演算子というものを使っています．これは，

　　　1式？　2式:　3式

の形式で与えられるもので，１式が真（０以外）ならば２式の値を，偽(０)ならば３式の値を全体の値とするものです．

再帰呼び出しのプロセスを追ってみると，たとえば４！を求める場合，関数は４を０と比べます．４＝０でないので，４＊ffact(３)の値が答えとなりますが，ffact(３)ではまた３を０と比べ，３＊ffact(２)を値とし，またffact(２)は２を０と比べ………というようになり，４＊ffact(３)→４＊(３＊ffact(2))→４＊３＊(２＊ffact(１))→４＊３＊２＊(1＊ffact(０))→４＊３＊２＊１＊１というプロセスで24という値を返してくるのです．

最大値を求める　関数の引数とポインタの利用

```
/* MAXIMUM NUMBER */

main()
{
     short max, inp;

     max = 0;

     do {
          printf("INPUT NUMBER  ");
          scanf("%d", &inp);
          checkmax(&max, &inp);
     } while (inp != 0);
     printf("MAXIMUM = %5d\n", max);
}
checkmax(pm, pi)
short *pm, *pi;
{
     if (*pm < *pi)
          *pm = *pi;
}
```

整数を入力していき，最後に0を入力するとそれまで入力した数の最大値が表示されます．なお入力できる整数は32767までです．

このプログラムでは値を入力する度にcheckmaxという自作の関数を呼び出して，maxよりinpが大きい場合はmaxにinpの値を代入しています．前にも説明した通り，関数への引数はその値だけしか渡さないため，呼ばれた側の関数から，呼んだ側の変数の値を変えることは不可能です．そのため&で変数のアドレスを渡すことにより，関数から値の変更がきくようにしてあるわけです．

checkmax関数が呼ばれる際には引数はアドレスとして渡されるため，引数宣言はポインタとして行われなければなりません．そして変数名に関係なく，関数内のポインタによって値が代入されるのです．

このように，何か変数の値に介入するような関数を作る場合は，ポインタを利用しなければ，何の意味もない関数を作ることになりかねません．

行列式の解　多次元配列

```
/* DETERMINANT */

main()
{
	short det;
	static short a[3][3] = {
		{1, 5, 3},
		{2, 3, 1},
		{2, 4, 3}
	};

	det = a[0][0]
		* (a[1][1] * a[2][2] - a[1][2] * a[2][1]);
	det -= a[0][1]
		* (a[1][0] * a[2][2] - a[1][2] * a[2][0]);
	det += a[0][2]
		* (a[1][0] * a[2][1] - a[1][1] * a[2][0]);
	printf("DETERMINANT = %5d\n", det);
}
```

3行3列の行列式を求めるプログラムです.

Cでの多次元配列は，1次元の後ろにまた大カッコをつなげて配列の大きさを宣言することによって使うことができます.

配列には，初期値を与えることが可能で，上記の例のように次元ごとに中カッコで分けて書いたり，また内側の中カッコを省略して書く場合もあります.

配列宣言のところのstaticとは，変数の記憶クラスを指定するものです. 記憶クラスとは，変数が持つ性質を決定するものです.

auto	(自動)
static	(静的)
extern	(外部的)
register	(レジスタ)

の4つがあり，通常はautoになっています. staticの指定をすると，その指定された関数の中でないと参照はできませんが，そのデータはずっと持ちつづけます. externではデータをずっと持ちつづけることはstaticと同じですが，そのデータは他の関数からでもexternの変数定義が行われていれば，参照することができます. registerは，その使っている機械のレジスタにおかれるため，移植性のよくないプログラムを作ってしまうおそれがあります. ただし直接レジスタに値を与えるのですから，演算スピードは速くなります.

月齢計算

構造体の応用

```
/* THE MOON'S AGE */

main()
{
	struct {
		short year;
		short month;
		short day;
	} date;

	short rev;
	float age;

	printf("YEAR = ");
	scanf("%d", &date.year);
	printf("MONTH = ");
	scanf("%d", &date.month);
	printf("DAY = ");
	scanf("%d", &date.day);

	switch(date.month) {
	case 1:
		rev = 1;
		break;
	case 2:
		rev = 2;
		break;
	case 3:
	case 5:
		rev = -1;
		break;
	default:
		rev = 0;
		break;
	}
	age = ((date.year - 1740.0) * 210.0 / 19.0 - 2.0
		+ date.month + date.day + rev) / 30.0;
	while (age > 1.0)
		age -= 1.0;
	age *= 30.0;
	printf("MOON'S AGE = %3.1f\n", age);
}
```

このサンプルプログラムは月齢（月の満ち欠けを表す日数）を計算するものです．プログラムを走らせると年，月，日をきいてくるので，順に入力するとその日付の月齢を計算して表示します．なお，「年」は西暦で1750年から2200年までの間のみ有効です．

簡単にプログラムを説明すると，まず日付の構造体date，修正値rev，月齢ageの各変数を宣言します．次のswitch文では月齢計算の修正値を求めています．そして，月齢計算を行い最後に表示をします（while文はageを１以下にするためです）．

使っている数式は次の通りです．

$$B=\frac{(Y-1740)\times 210\div 19-2+M+D+N}{30}$$

月齢＝｛B－INT(B)｝×30

Y：西暦年（1750～2200）

M：月

D：日

N：修正　１月→１，２月→２，3, 5→－1，その他の月は０

また，月齢の目安としては次を参考にしてください．

0：新月　3：三日月　7：上弦の月（半月）　15：満月

22：下弦の月（半月）　（30）　：新月

構造体は何らかの関連がある変数の集りに１つの名前を付け，まとめて扱かえるようにしたものです．これを使うと複雑なデータ構造やプログラムの流れをすっきりさせることができます．一般にレコードと呼ばれるデータ型式に使うことが多く，たとえばこのサンプルプログラムで使っている日付や名簿，図書目録などに，また，Ｃ言語ではシステムコールにそのマシンのレジスタの構造体を使っています．

構造体に許されているのは，そのアドレスを得ることと，そのメンバーをアクセスすることだけです．したがって構造体の代入や，関数への引き渡しは直接できません．そこで構造体のポインタがよく使われます．また構造体は入れ子にしてもよく，共用体と一緒に使っても構いません．さらに，ポインタを使えば自分と同じ型の構造体を参照することも可能です．これによりポインタでつながれたリスト構造を持たせることができます．

共用体およびビット・フィールドの考え方も構造体と一緒です．ただ，共用体はそのメンバーのオフセットが０であり，ビット・フィールドはそのメンバーを数ビットに圧縮したものです．

10進→16進変換　ループと配列

```
/* DECIMAL SYSTEM TO HEXADECIMAL SYSTEM */

main()
{
	static char data[16] =
		{'0', '1', '2', '3', '4', '5', '6', '7',
		 '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
	char hex[5];
	short dec, num, i;

	for (; ; ) {
		for (i = 0;.i <=3; i++)
			hex[i] = ' ';
		hex[4] = '\0';
		printf("DECIMAL = ");
		scanf("%d", &dec);
		if (dec == 0)
			break;
		for (i = 3; dec != 0; i--) {
			num = dec % 16;
			dec /= 16;
			hex[i] = data[num];
		}
		printf("HEXADECIMAL = %s\n\n", hex);
	}
}
```

10進数をそれに対応する16進数に変換するプログラムです．実行させると，“DECIMAL＝”と表示されて入力持ちとなりますから，何か10進数を入力してください．するとHEXADECIMAL＝”と，16進数が表示されます．0を入力すると終わります．

プログラムではまず配列を初期化して文字列を代入しています．この配列が16進数のデータとなるわけです．for(；；)は条件判断の項がないため無限ループとなります．ループの終了はbreak文によっています．次のforループは16進数の文字列を代入する配列を初期化します．printf関数で文字列を表示する場合，引数はその配列名となり．’/0’までの文字列を出力するのです．hex［4］＝’/0’；という文はそのために入れています．プログラムの流れから見れば，この文は一番初めにおき，ループの中に入れる必要はないのですが，こうした方が働きがわかりやすいと思い，ここに入れています．あとは入力された数値を16で割って余りを取り出す作業をくり返し16進数のデータを参照して16進数に変換しています．

16進→10進変換

関数strlenとindexの利用

```
/* HEXDECIMAL SYSTEM TO DECIMAL SYSTEM */

main()
{
     static char data[16] =
          {'0', '1', '2', '3', '4', '5', '6', '7',
           '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
     char hex[5], *index();
     short dec, fig, i, j, k;

     for (; ; ) {
          dec = 0;
          printf("HEXADECIMAL = ");
          scanf("%s", hex);
          if (hex[0] == '0')
               break;
          for (i = strlen(hex)-1, j = 0;
               i >= 0; i--, j++) {
               fig = index(data, hex[i]) - data;
               for (k = 0; k < j; k++)
                    fig *= 16;
               dec += fig;
          }
          printf("DECIMAL = %5d\n\n", dec);
     }
}
```

前のプログラムとは逆に16進数から10進数に変換するプログラムです．実行すると“HEXADECIMAL＝”と出力されますので4桁までで入力してください．英字は大文字で入力してください．“DECIMAL＝”と10進数に変換されて出力されます．0を入力すると終了です．

メインのループは前のプログラムと変わりありません．2つ目のループにあるstrlenという関数は文字列の長さを調べる標準ライブラリ関数で，引数は文字列の入っている配列の配列名です．

またindexという関数は文字列から文字を探す関数でその値は文字が入っているアドレスとなります．ですから第1パラメータに配列のポインタを引くことによって配列の添字が得られるのです．このプログラムでは，その添字を16進数の各桁の数字として利用し，桁の数だけ16を掛けて加えることで10進数に変換しています．

このプログラムも前のプログラムも，Cのプログラムとしてはあまりエレガントではありません．各自改良にチャレンジしてみてください．

左ロール

シフト演算子“<<”の使用

```
/* left roll */

#define SIZE sizeof(unsigned)*8

unsigned lroll(word, bit)
unsigned word;
unsigned char bit;
{
	unsigned tmp;

	tmp = word >> (SIZE - bit);
	word = word << bit;
	return(word | tmp);
}

main()
{
	unsigned word, lroll();
	unsigned char bit;

	printf("word(hexadecimal)?");
	scanf("%x", &word);
	printf("roll bit(decimal)?");
	scanf("%d", &bit);
	printf("%x(hexadecimal)\n", lroll(word, bit));
}
```

このサンプルプログラムは，通常はアセンブラが使われる左ロール命令をCで記述したものです．関数として作ってありますが，単体でも動作するようにメインルーチンを付けてあります．

左ロールする値（16進数）とロールするビット数（10進数）を入力して下さい．左ロールされた値（16進数）が表示されます．

ロールするビット数にintの範囲，すなわち10進数の16を超える数を入力すると表示する値は意味を持たなくなります．

プログラムは単純なものです．左にロールをさせるにはシフト演算子“<<”を使用すればよいのですが，単にそれだけでは右側のシフトされたビットが消えてしまいます．そこで，あらかじめ消えてしまうビットを右にシフトしておきます．このときunsignedと宣言してあるため最上位ビットから0がつめられます．最後に左シフトしたものとORをとり，関数値として返しています．

日数計算　変数としてのアンダースコア

```
/* THE NUMBER OF DAYS */

main()
{
	long s_year, s_month, s_day,
	     e_year, e_month, e_day,
	     daycount(), year, month, day;

	printf("START YEAR  ? ");
	scanf("%ld", &s_year);
	printf("      MONTH ? ");
	scanf("%ld", &s_month);
	printf("      DAY   ? ");
	scanf("%ld", &s_day);
	printf("  END YEAR  ? ");
	scanf("%ld", &e_year);
	printf("      MONTH ? ");
	scanf("%ld", &e_month);
	printf("      DAY   ? ");
	scanf("%ld", &e_day);

	printf("\n%ld DAYS\n", daycount(e_year, e_month,
	     e_day) - daycount(s_year, s_month, s_day));
}
long daycount(year, month, day)
long year, month, day;
{
	if (month >= 3)
	     month++;
	else {
	     year--;
	     month += 13;
	}
	day += month * 30.6;
	day += year * 365.25;
	day -= year / 100;
	day += year / 400;
	return(day - 428);
}
```

ある年月日からある年月日まで，何日あるのかを求めるプログラムです．実行すると，始めの年月日，終りの年月日をそれぞれ聞いてきますから入力してください．年は西暦で入力します．すると日数が計算されて表示されます．

目新らしいことは特にありませんが，'_'（アンダースコア）が変数の一部として使うことができることに注意してください．見やすいプログラムを作るためによく使われます．ただ'_'は単独では変数として使えません．

第3部

Cツール編

本編の読み方

本編はUNIX-likeなプログラミング・ツールを主体とした，ユーティリティプログラムのソースリストを公開したものです．

本編は各OS別に次の3部からなっています．

OS名	対象CPU	開発処理系
CP/M-80	Z80(8080)	AZTEC CII(ver.1.06)
MS-DOS	8086, 8088	OPTIMIZING C(ver.2.10), DeSmet C(ver.2.2)
OS-9	6809	MICROWARE C(ver.1.1.4)

本編ではUNIX-likeなシステムであるMS-DOSとOS-9ではUNIXコマンドの実現を，CP/MではCP/M+-likeおよびオリジナルなユーティリティコマンドの作成を目指しました．

ソースコードはすべてオリジナルであり，既製のOSのソースコードを利用したものではありません．また著作権は当会および技術評論社に帰属します．

依存する機種が明示してあるプログラム（たとえばグラフィック・パッケージ）以外は各OSにのみに依存しています．

各プログラムには，**Program Outline**で概略を，**Explanation**で具体的な使用法を説明してあります．また，**Attention**の欄があるプログラムは，コンパイル等に特別な注意が必要なものです．

正規式の使用法

今回，収録したツールの中にはUNIXのものを他のOSの上で実現したものがかなり含まれています．

これらのツールの大半はUNIXと同様に正規式（regular expression）による表現が扱えます．

正規式は簡単にいえば，特殊な意味を持った文字(メタキャラクタ)を使って複雑なことを簡略に表記するものです．

では，代表的なものを順を追って解説していきます．

①"^" と "$"

"^"（サーカムフレックス）は行の初めを表し，"$"（ダラー）は行末を表します．ただし，規定の位置にない場合は意味を失います．

例　^ABC ——— ABCで始まる行を表す
　　ABC$ ——— 文末がABCで終わる行を表す
　　A$BC ——— "A$BC"という文字列を表す
　　^$ ——— 空白行を表わす（これは頻繁に使われる）

②"."

"."（ピリオドは1対1で他の文字と対応します．ただし，空白やピリオド自身は考慮されません．また行単位で対応するため，2つの行にまたがっては対応しません．

例　A... ——— Aで始まる4文字の単語に対応する
　　^A... ——— 行の先頭にあるAで始まる4文字の単語に対応する
　　A..¥. あるいは A..＼. ——— Aで始まる3文字の単語とピリオドに対応する

③"[" と "]" と "-"

"[" と "]"（大カッコ）で囲まれた文字は対応すべき文字の集まりの規則を表しています．また "-"（ハイフン）でつなぐことによってASCII順に略記できます．

例　[Aa] ——— Aとaの双方に対応する
　　[A-Z] ——— AからZまでの文字に対応
　　[A-z] ——— すべての大文字と小文字との間に含まれる特殊文字に対応する（ASCIIコード表参照）

TINY EDITOR

短いファイルの作成のために

■Program Outline

SUBMITファイルなどの短いファイルを作成するのに，EDなどのエディタを使うのはめんどうなものです．とはいっても，PIPを使用するとCRとLFを別々に入力しなければなりません．そこで，短いファイルの作成のためにこのようなエディタを作ってみました．

■Explanation

COMファイル名はTED．COMとします．

コマンドラインからの入力は次のようになります．

A＞TED　FILENAME

この入力によって，FILENAMEのファイルがオープンされます．つづいてキーボードからテキスト行を入力します．このとき，1行の入力はCRによって終了することになります．すべての行を入力し終わったなら，ESC CRを入力することにより，エディットを終了して下さい．CRを入力しないかぎり，1行の範囲で何度でもエディットを行うことができます．

ファイルをセーブするにあたって，このエディタではディスク上に同名のファイルがあった場合，EDなどのようにディスク上のファイルをバックアップファイルとして保存するなどの安全策を講じていませんので，もとのファイルを消去してしまいます．ですから，ファイル作成の前にディスク上のファイル名を確認しておくように注意して下さい．

■Source List

```
/*                                    */
/*      Tiny editor    rev 1.0        */
/*                                    */
/*       Copyright (C) 1984           */
/*      Program   by   M.Hanari       */
/*                                    */

#include "stdio.h"———標準入出力ヘッダのインクルード

#define NULL      0
#define E_CODE   27     /* end of edit code (ESC) */
#define LN_BUF 100 ———1行の文字数を定義

main(argc,argv)
int argc;
char *argv[];
    {
    if (argc == 1)
        puts("Not file name !!");———ファイル名がないならエラー表示
    else
        edit(argv[1]); ———エディットを行う
    }

/*  file open and edit  */
edit(fl_name)
char *fl_name;
    {
    FILE *fp,*fopen();
    char a[LN_BUF]; ———aという文字列変数領域をとる

    if ((fp = fopen(fl_name,"w")) != NULL)—ライトファイルを
        {                                   オープンする
        while (*gets(a) != E_CODE) —キーボードからエスケープコードが
            fprintf(fp,"%s\n",a);     入力されると入力と出力をくり返す
        fclose(fp);——
        }          └——ファイルのクローズ
    }
```

LINE SAVER

プログラムやファイルの一部をセーブ

■Program Outline

このプログラムはソースプログラムやドキュメントファイルなどから，一部を抜き出してセーブするためのものです．C言語では，関数をライブラリとして保存する場合がよくありますが，この分割してセーブができる機能は，ライブラリの作成などに利用できるはずです．

■Explanation

COMファイル名はLSAV．COMとします．

コマンドラインからの入力は次のようになります．

A＞LSAV　FILENAME　範囲　FILENAME

パラメータはいずれも省略することはできません．なお，上記の"範囲"には次の3通りがあります．

ーn	先頭からn行まで
mーn	m行からn行まで
nー	n行から最後まで

例　A＞LSAV　TEST.C　100ー200　TEST.MD1

（TEST.Cの100行から200行までをTEST.MD1の名前でセーブ）

A＞LSAV　TEST.C　250ー　TEST.MD2

（TEST.Cの250行から最後までをTEST.MD2の名前でセーブ）

■Source List

```
/*                                     */
/*      Line saver      Rev 1.0        */
/*                                     */
/*       Copyright (C) 1984            */
/*      Program  by  M.Hanari          */
/*                                     */

#include "stdio.h" ―――標準入出力ヘッダのインクルード
#include "ctype.h" ―――文字処理に関するヘッダのインクルード

FILE *fp,*fopen();

main(argc,argv)
int argc;
char *argv[];
    {
    int i,j;
    unsigned int st,ed;
    char ad[10];

    if (argc == 3 || argc == 4)
        {
        if ((fp = fopen(argv[1],"r")) != NULL) ―リードファイルのオープン
            {
            for (i = 0,j = 0;
                     isdigit(ad[j] = argv[2][i]);i++,j++)―― adに
                ;                                  第2パラメータの最初を代入
            ad[j] = '\0';
            st = (st = atoi(ad)) ? st : 1;――adを数値に変換しstに代入.
            for (i++,  j = 0;                    もしstが0なら1に.
                     isdigit(ad[j] = argv[2][i]);i++,j++)―adに
                ;                                  第2パラメータの2番目を代入
            ad[j] = '\0';
            ed = (ed = atoi(ad)) ? ed : 65535;―adを数値に変換しedに
            if (argc == 3)                    代入. edが0なら65535を代入.
                ln_read(st,ed,"CON:"); ―ファイル名がなければコンソールに出力
            else
                ln_read(st,ed,argv[3]);―ファイル名があればそのファイルに出力
            }
        fclose(fp);――――ファイルのクローズ
        }
    else
        puts("parameter error"); ――――エラーを出力
    }

/*  1 line read & write   */
#define LIN_BUF 200

char buf[LIN_BUF];

ln_read(s,e,name)
unsigned int s,e;
char name[];
    {
```

```
    FILE *wfp;
    int ic,
        i = 1;

    if ((wfp = fopen(name,"w")) == NULL)—ライトファイルをオープンする
        return;
    while ((ic = getc(fp)) != EOF) ――――――ファイルから ic に 1 つ入力
        {
        if (!strcmp(name,"CON:"))――ファイル名が“CON:”ic に $1a を代入
            if (ic == 0x1a)
                {
                i++;
                continue;
                }
        buf[0] = ic;
        fgets(&buf[1],LIN_BUF - 1,fp);――――――ファイルから1ライン入力
        if (i >= s && i <= e)
            fputs(buf,wfp); ――――――ファイルにバッファの内容を出力
        i++;
        }
    fclose(wfp); ――――――ライトファイルのクローズ
    }

/* ascii to integer */
atoi(cp)
register char *cp;
    {
    register unsigned i;
    register sign;

    while (*cp == ' ' || *cp == '\t')――タブかスペースならスキップする
        ++cp;
    sign = 0;
    if ( *cp == '-' ) ――――――もし，マイナスならサインフラグを ON
        {
        sign = 1;
        ++cp;
        }
    else
        if ( *cp == '+' ) ――――――もし，プラスならスキップ
        ++cp;

    for ( i = 0 ; isdigit(*cp) ; ) ┐
        i = i*10 + *cp++ - '0';    ┘――――――文字を数値に変換する
    return sign ? -i : i; ――――――符号の処理
    }
```

DUMP PLUS

各種のデバイスにダンプを出力

■Program Outline

このプログラムはCP/MのDUMPコマンドにアスキーダンプの機能を加えたものです．さらに，そのダンプをCRTのほかに，プリンタに出力することもできるうえに，ファイルとして出力することもできます．

■Explanation

COMファイル名はDUMP.COMとします．

さて，コマンドラインからの入力は次のようになります．

A＞DUMP　FILENAME　〔OUTPUT〕

この〔OUTPUT〕には，出力先としてのファイル名やデバイス名を指定できます．無指定のときには，CRT（CON：）に出力します．

例　A＞DUMP　TEST.COM　LST：

（TEST.COMをプリンタに出力）

A＞DUMP　TEST.COM　TEST.DMP

（TEST.COMをTEST.DMPのファイル名でディスクに出力）

出力例

```
A>DUMP TEST.COM
0000 : C3 26 12 11 F9 FF CD 8A 26 21 13 00 39 5E 23 56   .&..y...&!..9^#V
0010 : 21 01 00 CD F5 27 CA 02 02 21 13 00 39 5E 23 56   !...u'...!..9^#V
0020 : 21 02 00 CD 21 28 CA 7C 01 21 01 00 EB 21 08 00   !...!(.|.!...!..
0030 : 39 73 23 72 C3 46 01 21 08 00 39 E5 7E 23 66 6F   9s#r.F.!..9.~#fo
0040 : 23 EB E1 73 23 72 21 13 00 39 5E 23 56 EB 2B E5   #..s#r!..9^#V.+.
0050 : 21 0A 00 39 5E 23 56 E1 CD 20 28 CA 7C 01 21 08   !..9^#V.. (.|.!.
0060 : 00 39 5E 23 56 EB 29 EB 21 15 00 39 7E 23 66 6F   .9^#V.).!..9~#fo
0070 : 19 5E 23 56 D5 CD 11 02 D1 C3 37 01 21 13 00 39   .^#V......7.!..9
0080 : 5E 23 56 EB 2B 29 EB 21 15 00 39 7E 23 66 6F 19   ^#V.+).!..9~#fo.
0090 : 5E 23 56 D5 21 28 2E E5 CD C7 24 D1 D1 21 03 02   ^#V.!(....$..!..
00A0 : E5 21 28 2E E5 CD 25 0C D1 D1 22 37 2E 7C B5 CA   .!(...%..."7.|5.
00B0 : 02 02 2A 9C 28 7C B5 CA CB 01 21 05 02 E5 21 0C   ..*(|5...!...!.
00C0 : 00 39 E5 CD C7 24 D1 D1 C3 D9 01 21 0A 02 E5 21   .9...$.....!...!
00D0 : 0C 00 39 E5 CD C7 24 D1 D1 21 0F 02 E5 21 0C 00   ..9...$..!...!..
00E0 : 39 E5 CD 25 0C D1 D1 22 39 2E 7C B5 CA FA 01 CD   9..%..."9.|5.z..
00F0 : 6F 03 2A 39 2E E5 CD 83 0F D1 2A 37 2E E5 CD 83   o.*9.....*7...
0100 : 0F D1 C9 72 00 43 4F 4E 3A 00 4C 53 54 3A 00 77   ...r.CON:.LST:.w
```

■Source List

```
/*                                  */
/*    Dump plus      Rev 1.0        */
/*                                  */
/*     Copyright (C) 1984           */
/*    Program  by  M.Hanari         */
/*                                  */

#include "stdio.h"  ―――標準入出力ヘッダのインクルード
#include "ctype.h"  ―――文字処理ヘッダのインクルード

#define NULL 0 ―――ヌルを0と定義する
#define STD_DEV "con:"―――出力デバイスのデフォルトを"con:"とする

main(argc,argv)
int argc;
char *argv[];
     {
     char dev[20]; ―――devという文字列変数領域をとる

     strcpy(dev,STD_DEV); ―――文字の代入
     if (argc != 1)
          {
          if (argc == 3)
               strcpy(dev,argv[2]);―――デバイス名の指定あれば代入
          dump(argv[1],dev);  ―――ファイルのダンプを行う
          }
     }

/*  file open and read   */
int buf[20];

dump(r_name,w_name)
char *r_name,*w_name;
     {
     FILE *fp_r,*fp_w,*fopen();
     int i;
     unsigned int adrs;

     if ((fp_r = fopen(r_name,"r")) == NULL) ―――リードファイルをオープン
          return; ―――エラーなら終了
     if ((fp_w = fopen(w_name,"w")) != NULL) ―――ライトファイルをオープン
          {
          adrs = 0; ―――アドレス表示用度数の初期化
          while ((buf[0] = getc(fp_r)) != EOF) ―――リードファイルを
               {                                  全部読んだら終了
               for (i = 1;i < 16;i++)
                    buf[i] = getc(fp_r); ―――バッファにファイルの内容をためる
               hex_dump(adrs++,fp_w);―――バッファの内容をHexにダンプ
               ascii_dump(fp_w); ―――バッファの内容をアスキーダンプ
               }
          fclose(fp_w); ―――ライトファイルのクローズ
          }
     fclose(fp_r); ―――リードファイルのクローズ
     }
```

```
/*  hex dump output  */
hex_dump(add,fp)
unsigned int add;
FILE *fp;
    {
    int i;
    char s[10];

    itoh(add,s); ──────add の内容を16進の文字に変換
    fprintf(fp,"%03s0 : ",s); ──────アドレスを出力
    for (i = 0;i < 16;i++)
        {
        itoh(buf[i],s); ──────バッファの内容を16進の文字列に変換

        fprintf(fp,"%02s ",s); ──────16進変換したバッファを出力
        }
    }

/*  ascii dump output  */
ascii_dump(fp)
FILE *fp;
    {
    int i;

    fprintf(fp,"  "); ──────空白を出力
    for (i = 0;i < 16;i++)
        putc(isprint(buf[i]) ? buf[i] : '.' ,fp); ──┐
    fprintf(fp,"\n"); ──────改行を出力              │
    }                                              バッファの内容を出力.
                                                   もしコントロールコー
/*  integer to hex-string  */                      ドならピリオド(.)を出力
itoh(n,s)
char *s;
unsigned int n;
    {
    char *itoh_s();

    *(itoh_s(n,s)) = '\0'; ──────＊Ｓの後にヌルコードを付加
    }

char *itoh_s(n,s)
char *s;
unsigned int n;
    {
    if ((n/16) > 0)
        s = itoh_s(n/16,s); ──────itoh _ s の再帰呼び出し
    *s++ = ((n % 16 > 9) ? n % 16 - 10 + 'A' : n % 16 + '0');
    return(s);
    }
```

EXPAND DIR

ワイルドカードが使えるDIRコマンド

■Program Outline

このExpand Dirは，ワイルドカードを使えるようにする関数を作成し，その使用例としてDIRコマンドに属性を表示する機能を加えたものです．

■Explanation

COMファイル名はEXDIR.COMとします．

このExpand Dirは通常のDIRコマンドと同様に使用できます．コマンドラインからの入力は次のようになります．

A>EXDIR FILENAME

ここで、FILENAMEにはワイルドカードが使用できます．

表示画面上では，システム属性のファイルはファイル名の後に閉じカッコ“)”が，書き込み禁止属性のファイルには“＊”が付きます．

出力例

```
A>EXDIR
A:TURBO    .COM *) A:TURBO    .OVR *) A:STAT     .COM *  A:PIP      .COM *
A:POWER    .COM *  A:ZSID     .COM *  A:SUBMIT   .COM *  A:DUMP     .COM *
A:COPY     .COM *  A:TED      .COM *  A:PSUB     .COM *  A:TY       .COM *
A:PLST     .COM *  A:AZTEC/C  .    *  A:LOOK     .C      A:TYPE     .BAK
A:BENCH1   .BAK    A:BENCH1   .ASM    A:BENCH1   .C      A:BENCH1   .O
A:BENCH1   .COM    A:EXDIR    .COM
```

ワイルドカードの関数はwild（in_name, out_name）の形でコールされます．このとき，in_nameはファイル名の入った配列の先頭のポインタで，out_nameは関数wild( )によってサーチされたファイルネームが入る配列の，先頭のポインタです．また，wild( )自体は対応するファイルがあれば値として0を持ち，ないときには－1（EOF）の値を持ちます．

関数wild( )は，in_nameに対応するファイルをディレクトリよりサーチし，対応するファイルのディレクトリの32バイトをout_nameのポインタから代入します．したがって関数wild( )を使用するにあたって，out_name用の配列(キャラクタ32バイト)を用意する必要があります．ファイルネームのみを取り出す方法は，ソースリストの関数get_nm( )を参考にして下さい．

in_nameにワイルドカードの含まれる場合は，1つのファイルネームに対して複数のファイルが対応しますので，通常は次のような形でwild( )をコールして下さい．

```
while (wild(in_name,out_name) != EOF)
        {
         処理
        }
```

エラーメッセージ

AZTEC CII では標準リンクモジュールとして，C.LIB（.COMファイルを生成する），R.LIB（R.COMと合わせて使う.OVRファイルを生成する),T.LIB（関数の機能を限定して，サイズを短縮した.COMファイルを生成する），M.LIB（float型の変数に対する演算用の関数等を含む数値演算ライブラリ）がありますが，リンクするときにリンクすべきファイルを正しい順序で指示する必要があります．

たとえば，example.o（プログラム中でfloat型の変数を扱っている）というファイルからexample.comというファイルを生成する場合，リンクは次のような順序で入力する必要があります．

A＞<u>ln example. o m. lib c. lib</u>

上記の例ではm. libとc. libを逆にすると次のようなメッセージを返してリンクを中断してしまいます．

Undefined simbol _××××.

■Source List

```
/*                              */
/*    Expand dir   Rev 1.0      */
/*                              */
/*     Copyright (C) 1984       */
/*    Program  by  M.Hanari     */
/*                              */

#include "stdio.h"

main(argc,argv)
int argc;
char *argv[];
    {
    char carg[20],
         fc_nm[32],
         name[20],
         r_at,
         s_at;
    int ct = 0;

    strcpy(carg,argc == 1 ? "*.*" : argv[1]);
    if (strlen(carg) == 2 && carg[1] == ':')
        strcat(carg,"*.*");
    while (wild(carg,fc_nm) != EOF)
        {
        r_at = fc_nm[ 9] & 128 ? '*' : ' ';
        s_at = fc_nm[10] & 128 ? ')' : ' ';
        get_nm(fc_nm,name);
        printf("%s %c%c  ",name,r_at,s_at);
        if (ct++ % 4 == 3)
            putchar('\n');
        }
    }

get_nm(in,out)
char *in,*out;
    {
    int i;

    out[0] = in[0] + 'A' -1;
    out[1] = ':';
    for (i = 1;i < 9;i++)
        out[i+1] = in[i];
    out[10] = '.';
    for (i = 9;i < 12;i++)
        out[i+2] = in[i] & 0x7f;
    out[14] = '\0';
    }
```

```
/*  wild_card function  */ ——————————関数wild( )
#define DMA_ADR 0x80

wild(in_name,ret_name)
char in_name[],
     *ret_name;
    {
    char wfcb[40];
    static char res_name[20];
    static int ct;
    char *pc;
    int i,
        rp;

    if (strcmp(in_name,res_name))
        {
        strcpy(res_name,in_name);
        ct = 0;
        }
    bdos(26,DMA_ADR);
    fcbinit(in_name,wfcb);
    rp = bdos(17,wfcb);
    if (ct > 0)
        for (i = 0;i < ct;i++)
            if ((rp = bdos(18,0)) == 255)
                break;
    if (rp == 255)
        {
        *res_name = 0;
        return(-1);
        }
    pc = rp * 32 + DMA_ADR;
    ret_name[0] = (*wfcb == 0) ? bdos(25,0)+1 : *wfcb;
    for (i = 1;i < 32;i++)
        ret_name[i] = pc[i];
    ct++;
    return(0);
    }
```

RENAME PLUS wild( )の応用

ワイルドカードの使えるRENAME

■Program Outline

CP/M V.2.2では不可能な複数のファイルを，ワイルドカードを使用して一度にRENAMEすることができます．このプログラムは，関数wild( )を応用したものです．

■Explanation

COMファイル名はRENAME. COMとします．

コマンドラインからの入力は，次のように通常のリネーム(REN)と同じようになります．

A＞RENAME NEWFILE = OLDFILE〔N〕

(N：実行の確認をしない)

コマンドを実行するとファイル名を表示し，リネームの確認をとります．ただし，Nオプション指定時は無条件にリネームします(**出力例**参照)．

このプログラムにおいて，関数wild( )は一部変更されています．wild( )内の6行目のstatic int ct;を削除しています．

出力例

```
A>RENAME NAME.*=RENAME.*
A:RENAME  .C   to A:NAME    .C   (y/n)N
A:RENAME  .BAK to A:NAME    .BAK (y/n)Y
A:RENAME  .COM to A:NAME    .COM (y/n)N
```

注）Y,Nはコンソールから入力

■Source List

```
/*                                     */
/*     Rename plus    Rev 1.0          */
/*                                     */
/*      Copyright (C) 1984             */
/*     Program  by  M.Hanari           */
/*                                     */

#include "stdio.h"

int ct;

main(argc,argv)
int argc;
char *argv[];
    {
    char f_name[32],
         carg[20],
         name[40],
         new_nm[20],
         w_name[20],
         c;
    int  q_mod = 0,
         i,j;

    if (argc == 3 && argv[2][0] == 'N')
        q_mod++;
    if (argc != 1)
        {
        for (i = 0;(w_name[i] = argv[1][i]) != '=';i++)
            ;
        w_name[i++] = '\0';
        for (j = 0;(carg[j] = argv[1][i]) != '\0';i++,j++)
            ;
        fcbinit(w_name,name);
        get_nm(name,w_name);
        while (wild(carg,f_name) != EOF)
            {
            get_nm(f_name,name);
            strcpy(new_nm,w_name);
            conv_nm(name,new_nm);
            printf("%s to %s ",name,new_nm);
            if (q_mod)
                putchar('\n');
            else
                {
                printf("(y/n)");
                while ((c = toupper(getchar())) != 'Y'
                                         && c != 'N')
                ;
                }
            if (q_mod || (!q_mod && c == 'Y'))
                {
                if (rename(name,new_nm) == EOF)
                    {
```

```
                        puts("error");
                        exit(0);
                        }
                    ct--;
                    }
                }
            }
        }

/*  get file_name   */
get_nm(in,out)
char *in,*out;
    {
    int i;

    out[0] = in[0] + 'A' - 1;
    out[1] = ':';
    for (i = 1;i < 9;i++)
        out[i+1] = in[i];
    out[10] = '.';
    for (i = 9;i < 12;i++)
        out[i+2] = in[i] & 0x7f;
    out[14] = '\0';
    }

/*  convert new name   */
conv_nm(old,new)
char *old,*new;
    {
    int i;

    new[0] = old[0];
    new[1] = old[1];
    for (i = 2;i < 14;i++)
        if (new[i] == '?')
            new[i] = old[i];
    }

/*  wild_card function   */
#define DMA_ADR 0x80

wild(in_name,ret_name)
char in_name[],
     *ret_name;
    {
    char wfcb[40];
    static char res_name[20];
    char *pc;
    int i,
        rp;

    if (strcmp(in_name,res_name))
        {
        strcpy(res_name,in_name);
        ct = 0;
        }
```

```
bdos(26,DMA_ADR);
fcbinit(in_name,wfcb);
rp = bdos(17,wfcb);
if (ct > 0)
    for (i = 0;i < ct;i++)
        if ((rp = bdos(18,0)) == 255)
            break;
if (rp == 255)
    {
    *res_name = 0;
    return(-1);
    }
pc = rp * 32 + DMA_ADR;
ret_name[0] = (*wfcb == 0) ? bdos(25,0)+1 : *wfcb;
for (i = 1;i < 32;i++)
    ret_name[i] = pc[i];
ct++;
return(0);
}
```

TYPE PLUS

ワイルドカードの使えるCP/M PLUSライクのTYPEコマンド

■Program Outline

このプログラムにより，TYPEコマンドの従来の機能の他に，ワイルドカードを使うことができます．CP/M PLUSのようにファイルネームを指定する場合には，？と＊の両方のワイルドカードの使用を可能にしています．さらに，プリンタへの出力もできるようになっています．ただし，CP/M V.2.2では，このワイルドカードの使用も，プリンタへの出力もできません．

■Explanation

COMファイル名はTYP.COMとします．

コマンドラインからの入力は次のとおりです．

A＞TYP.COM　FILENAME〔オプション〕

このとき，ワイルド(＊)を使用した場合は複数のファイルネームを指定できますが，ワイルドを使わないとき以外は，ファイルネームを1つしか指定することはできません．オプションは**表1**参照．

このコマンドの実行により，出力は1ページごとにストップします(**出力例1**)．Nオプションの指定がない場合はモードに関係なく連続してプリンタに出力します．なお，ファイルが複数のときはファイルごとに次のページに送ります(**出力例2**)．

表1

オプション	機　　能
N	1ページごとにストリップしない
P	プリンタに出力する

出力例1（ファイルを1ページごとに出力）

```
     A:TYP      .C
/*                               */
/*     Type plus     Rev 1.0     */
/*                               */
/*      Copyright (C) 1984       */
/*     Program  by  M.Hanari     */
/*                               */

#include "stdio.h"

#define CPM_EOF 0x1a

char dev[2][5] = {"CON:","LST:"};

int  md = 0,
     pg = 0,
     ln = 0;

main(argc,argv)
int argc;
char *argv[];
    {
    FILE *fp,*fpw;
Press RETURN to Continue
```

出力例2（複数のファイルのときはファイルごとにページを送る）

```
    if (ct > 0)
        for (i = 0;i < ct;i++)
            if ((rp = bdos(18,0)) == 255)
                break;
    if (rp == 255)
        {
        *res_name = 0;
        return(-1);
        }
    pc = rp * 32 + DMA_ADR;
    ret_name[0] = (*wfcb == 0) ? bdos(25,0)+1 : *wfcb;
    for (i = 1;i < 32;i++)
        ret_name[i] = pc[i];
    ct++;
    return(0);
    }

A:TYP      .C
/*                               */
/*     Type plus     Rev 1.0     */
/*                               */
/*      Copyright (C) 1984       */
/*     Program  by  M.Hanari     */
Press RETURN to Continue
```

■Source List

```
/*                              */
/*     Type plus     Rev 1.0    */
/*                              */
/*      Copyright (C) 1984      */
/*     Program  by  M.Hanari    */
/*                              */

#include "stdio.h"

#define CPM_EOF 0x1a

char dev[2][5] = {"CON:","LST:"};

int  md = 0,
     pg = 0,
     ln = 0;

main(argc,argv)
int argc;
char *argv[];
    {
    FILE *fp,*fpw;
    char f_name[32],
         name[20];
    char c;
    int  i = 0;

    if (argc == 3)
        while ((c = argv[2][i++]) != '\0')
            if (c == 'N')
                pg = 1;
            else
                if (c == 'P')
                    {
                    md = 1;
                    pg = 1;
                    break;
                    }
    if (argc != 1)
        {
        if ((fpw = fopen(dev[md],"w")) == NULL)
            puts("device error");
        else
            {
            while (wild(argv[1],f_name) != EOF)
                {
                get_nm(f_name,name);
                if ((fp = fopen(name,"r")) == NULL)
                    {
                    puts("file read error");
                    exit();
                    }
                else
                    {
```

```
                    if (md)
                        putc('\f',fpw);
                    fprintf(fpw,"%s\n",name);
                    page();
                    while ((i = getc(fp)) != EOF
                                && i != CPM_EOF)
                        {
                        putc(i,fpw);
                        if (i == '\n')
                            page();
                        }
                    fclose(fp);
                    putc('\n',fpw);
                    page();
                    }
                }
            fclose(fpw);
            }
        }
    }

/*  page mode process  */
page()
    {
    if (!pg && ln++ == 22)
        {
        printf("Press RETURN to Continue ");
        while (getchar() != '\n')
            ;
        ln = 0;
        }
    }

/*  get file_name  */
get_nm(in_ptr,out_ptr)
char *in_ptr;
char *out_ptr;
    {
    int i;

    out_ptr[0] = in_ptr[0] + 'A' - 1;
    out_ptr[1] = ':';
    for (i = 1;i < 9;i++)
        out_ptr[i+1] = in_ptr[i];
    out_ptr[10] = '.';
    for (i = 9;i < 12;i++)
        out_ptr[i+2] = (in_ptr[i] & 0x7f);
    out_ptr[14] = '\0';
    }
```

注）このソースリストに「EXPAND DIR」のところで使用した関数wild( )をリンクしてください.

SUBMIT PLUS

SUBMITコマンドの機能のアップ

■Program Outline

SUBMITコマンドは便利なコマンドですが，あまり多くの機能を持ってはいません．そこで，このプログラムにより，SUBMITコマンドの持つ機能以外に，IF文による条件分岐や文字列の出力，キー入力の機能を可能にしています．これらの機能によって，今までは処理に応じて複数のSUBMITファイルを必要としていたものを，1つのファイルで処理することができるようになります．

■Explanation

COMファイル名はPSUB.COMとします．プログラムの使用に先立って，コマンドファイル（SUBMITファイルに相当するもの）を作成しますが，このコマンドファイル内では表1の命令が使用できます．コマンドおよびIF文における＝（イコール）などは，必ず1つ以上の空白で区切って下さい．

$1，$2などは通常のSUBMITと同じようにコマンドラインからの文字列が代入され，＃0，＃1などは内部変数でKEY命令により各々1文字の値を持ちます．

IF文のネストは許可されず，そのため，IF～ELSE……IF～ELSEのようなときには，ELSEは直前にあるIFのみに対応します．条件部は（　）で囲み，IF文の終了には必ずENDIFを付けて下さい．

例
```
KEY  ＃1
IF  (＃1  =  "1")
    EX  "DIR  A:"  ────────  ＃1  =  "1"のとき実行
ELSE  IF  (＃1  =  "2")
    EX  "DIR  B:"  ────────  ＃1  =  "2"のとき実行
ELSE
    EX  "DIR  C:"  ────────  ＃1  <>  "2"のとき実行
```

実行は通常のSUBMITと同じように行います．

例　A＞PSUB　TEST

表1

コ　マ　ン　ド	機　　　能
EX　"～"	" " 内の命令を実行する
PRINT　"～"	" " 内の文字を画面に出力する
KEY　＃n	変数＃nに1文字入力
IF　(～)　～	(　)内が真のとき以下の文を実行
ELSE　～	直前のIF文の条件が偽のとき以下を実行
ENDIF	IF文の範囲の終了

注）IF文の条件は；＝(等しい)，！(等しくない)の2種類で変数＝定数の形をとり，変数には＃n，＄nが対応し，定数には "" で囲まれた文字列が対応します．

コマンドファイルの例

例1

```
PRINT "1:DIR"
PRINT "2:STAT"
PRINT "Select 1 or 2 ?"
KEY #1
IF (#1 = "1")
  EX "DIR"
ELSE IF (#1 = "2")
  EX "STAT"
ENDIF
```

例2

```
PRINT "1:Compile"
PRINT "2:Assemble"
PRINT "3:Link 'C.LIB'"
PRINT "4:Compile to Link 'C.LIB'"
PRINT "Select 1 to 4 ?"
KEY #1
IF (#1 = "1")
  EX "CC $1.C"
ELSE IF (#1 = "2")
  EX "AS $1.ASM"
ELSE IF (#1 = "3")
  EX "LN $1.O C.LIB"
ELSE IF (#1 = "4")
  EX "CC $1.C"
  EX "AS $1.ASM"
  EX "LN $1.O C.LIB"
ENDIF
```

■Source List

```
/*                                */
/*    Submit plus    Rev 1.0      */
/*                                */
/*     Copyright (C) 1984         */
/*    Program  by  M.Hanari       */
/*                                */

#include "stdio.h"
#include "ctype.h"

#define PASS  0
#define WRITE 1
#define EXE 2
#define PRT 3
#define KIN 4
#define IFF 5
#define ELE 6
#define ENF 7

#define MAX_LINE 80
#define MAX_COL  80

FILE *fp_r,*fp_w;

main(argc,argv)
int argc;
char *argv[];
    {
    FILE *fopen();

    if (argc != 1)
        if ((fp_r = fopen(argv[1],"r")) != NULL)
            {
            if ((fp_w = fopen("A:$$$.SUB","w")) != NULL)
                {
                mk_file(argc,argv);
                fclose(fp_w);
                }
            fclose(fp_r);
            }
    }

/*  file read and write $$$.sub  */
mk_file(argc,argv)
int argc;
char *argv[];
    {
    int i,ln_ct = 0;
    long l;
    char buf[80];
    char com_buf[MAX_LINE][MAX_COL];

    while ((i = char_in(argc,argv)) != EOF)
        if (i == WRITE)
            {
```

```
            skip('\"');
            for (i = 0;(buf[i] = getc(fp_r)) != '\"';i++)
                ;
            buf[i] = '\0';
            chg_para(argc,argv,buf);
            strcpy(com_buf[ln_ct++],buf);
            }
    for (l = 0;l < ln_ct;l++)
        {
        fseek(fp_w,l * 128L,0);
        putw(strlen(com_buf[ln_ct-l-1]),fp_w);
        fseek(fp_w,-1L,1);
        fputs(com_buf[ln_ct-l-1],fp_w);
        putc('\0',fp_w);
        }
    }

/*  command process   */
char word[40];
char var[10];

char_in(argc,argv)
int argc;
char *argv[];
    {
    static int ex_mod = 1;
    char c;

    if (word_in() == EOF)
        return(EOF);
    x_upper(word);
    switch(com_chk())
        {
        case EXE:                               /*    EX     */
            if (ex_mod == 1)
                return(WRITE);
            skip('\"');             /*  skip not ex-com   */
            skip('\"');
            break;
        case PRT:                               /*   PRINT   */
            skip('\"');
            putchar('\n');
            while ((c = getc(fp_r)) != '\"')
                putchar(c);
            break;
        case KIN:                               /*  KEY IN   */
            word_in();
            if (word[0] == '#')
                key_in();
            break;
        case IFF:                               /*    IF     */
            skip('(');
            word_in();
            ex_mod = (word[0] == '$') ?
                 test_str(argc,argv) : test_ch();
            skip(')');
```

```
            break;
        case ELE:                               /*   ELSE   */
            ex_mod = 1 - ex_mod;
            break;
        case ENF:                               /*   ENDIF  */
            ex_mod = 1;
            break;
        }
        return(PASS);
    }

/*  get word  */
word_in()
    {
    int ic,i;
    char c;

    while (isspace(ic = getc(fp_r)))
        ;
    if (ic == EOF)
        return(EOF);
    word[0] = ic;
    i = 1;
    while (! isspace(ic = getc(fp_r)))
        {
        if (ic != EOF)
            word[i++] = ic;
        else
            {
            word[i] = '\0';
            return(EOF);
            }
        }
    word[i] = '\0';
    return(NULL);
    }

/*  change upper case  */
x_upper(ptr)
char *ptr;
    {
    for ( ;*ptr != '\0';ptr++)
        *ptr = toupper(*ptr);
    }

/*  command check  */
com_chk()
    {
    if (! strcmp(word,"EX"))
        return(EXE);
    if (! strcmp(word,"PRINT"))
        return(PRT);
    if (! strcmp(word,"KEY"))
        return(KIN);
    if (! strcmp(word,"IF"))
        return(IFF);
```

```
    if (! strcmp(word,"ELSE"))
        return(ELE);
    if (! strcmp(word,"ENDIF"))
        return(ENF);
    return(PASS);
    }

/*  change $n to string  */
chg_para(argc,argv,o_buf)
int argc;
char *argv[],o_buf[];
    {
    char c_buf[80];
    char c;
    int i,j,k,str_no;

    k = 0;
    for (i = 0;o_buf[i] != '\0';i++)
        {
        if (o_buf[i] == '$')
            {
            str_no = o_buf[++i] - '0';
            if (argc < str_no + 2)
                {
                puts("parameter error");
                exit(1);
                }
            for (j = 0;(c = argv[str_no + 1][j]) != '\0';j++)
                c_buf[k++] = c;
            }
        else
            c_buf[k++] = o_buf[i];
        }

    c_buf[k] = '\0';
    strcpy(o_buf,c_buf);
    }

/*  string compare  */
test_str(argc,argv)
int argc;
char *argv[];
    {
    char cp_buf1[40],cp_buf2[40];
    char op;
    int i;

    strcpy(cp_buf1,word);
    chg_para(argc,argv,cp_buf1);
    word_in();
    op = word[0];
    skip('\"');
    for (i = 0;(cp_buf2[i] = getc(fp_r)) != '\"';i++)
        ;
    cp_buf2[i] = '\0';
    chg_para(argc,argv,cp_buf2);
```

```
        i = (strcmp(cp_buf1,cp_buf2) == 0);
        return(op == '=' ? i : 1 - i);
        }

/*  character compare  */
test_ch()
        {
        int i;
        char op;

        i = word[1] - '0';
        word_in();
        op = word[0];
        skip('\"');
        i = ((var[i] - getc(fp_r)) == 0);
        skip('\"');
        return(op == '=' ? i : 1 - i);
        }

/*  skip to mark  */
skip(mark)
char mark;
        {
        while (getc(fp_r) != mark)
                ;
        }

/*  get key  */
key_in()
        {
        char key_buf[20];

        gets(key_buf);
        var[word[1] - '0'] = toupper(*key_buf);
        }
```

FILE MARKER

関数名や変数名をファイルから探し出す

■Program Outline

ソースファイルの中から，単語(最高で10語まで)を探し出します．そして，ディスティネーションファイル(この場合はオブジェクトファイル)にソースファイルをコピーして，その行の下に単語のマーク0～9を，それぞれの単語の先頭に付けます．

このプログラムは，C言語のソースファイルから変数や関数が使われている場所をみつける場合などに有効です．

■Explanation

COMファイル名はMARC.COMとします．以下操作手順を示します．下線部が入力部分になっています．

①プログラム名を入力する．

MARK <CR>　　　(<CR>はリターンキー)

②ソースファイル名を入力する．

Source file name = ?
text.c <CR>

③ディスティネーションファイル名を入力する．

Distination file name = ?
text.mrk <CR>

④探したい単語を入力する．終わるときは，<CR>のみを入力する．

WORD(0)=?　ABC <CR>

WORD(1)=?　abc 〈CR〉

WORD(2)=?　〈CR〉

単語を10語入力し終わるか，あるいは〈CR〉だけを入力すると，次のようなさまざまなメッセージを出力して単語だけを探し始めます．

Source　＝text. c

Distination　＝text. mrk

Word number　＝1

　WORD(0)＝|word1|

　WORD(1)＝|word2|

プログラムの実行結果

```
A>mark
Source file name = ?
example.c
Distination file name = ?
example.mk
WORD(0)=?define
WORD(1)=?MAXLINE
WORD(2)=?maxline
WORD(3)=?{
WORD(4)=?(
WORD(5)=?

Source =example.c
Distination =example.mk
Word number = 4
   Word(0)=|define|
   Word(1)=|MAXLINE|
   Word(2)=|maxline|
   Word(3)=|{|
   Word(4)=|(|
```

このプログラムでは，探す単語を構成する文字は入力されたときの文字コードで比較されるので，大文字と小文字は異なる文字とみなされます．また，タブについては1文字として扱うためマークの位置がずれてしまうため，あらかじめ何文字かの空白文字（スペース）に置換しておく必要があります．スペースは他の文字と同様に1文字として扱います．

逆にスペースを上手に使うことによって，不必要な単語をマークしないですみます．なお，〈CR〉(キャリッジリターン）は単語入力の終了コードとして扱っているため，単語を構成する文字としては扱えません．

出力例

```
A>type example.mk
/*----- example program source -----*/
______________________________________

#define MAXLINE 10
_0________1_________

char string[30] = "abcDEFMAXLINEmaxlineMAX LINE";
_________________________1______2_________________

main()
_____4_
      {
_____3
      printf("Defined string is /%s/.\n", string);
___________4_____________________________________
      }
_____
```

■Source List

```
/*                                      */
/*     File marker    Rev 1.0           */
/*                                      */
/*      Copyright (C) 1984              */
/*     Program by M.Matsuzaki           */
/*                                      */

#include "b:stdio.h"

#define MAXWORD  10
#define WORDLEN  10
#define FNAMELEN 20
#define MAXLINE  150
#define EOD      0x1A
#define NOERR    0
#define EQU      0

/*  keyword table (struct)  */
struct w_table {
               int word_no;
               char word[MAXWORD][WORDLEN];
               };

main()
    {
    char              s_file[FNAMELEN],
                      d_file[FNAMELEN];
    struct w_table words;

    fname_in(s_file, d_file);
    word_in(&words);
    mesage(s_file, d_file, &words);
    mark(s_file, d_file, &words);
    }

/*  file name input  */
fname_in(s_file, d_file)
char *s_file,
     *d_file;
    {
    printf("Source file name = ?\n");
    gets(s_file);

    printf("Distination file name = ?\n");
    gets(d_file);
    }

/*  keyword input  */
word_in(words)
struct w_table *words;
    {
    int  no;
    char w_buf[WORDLEN];

    for (no = 0; no < MAXWORD; no++)
```

```
        {
        printf("WORD(%d)=?", no);
        gets(words->word[no]);
        if (*(words->word[no]) == '\0')
            break;
        }
    words->word_no = no - 1;
    }

/*  source file,distination file & keyword list out  */
mesage(s_file, d_file, words)
char                *s_file,
                    *d_file;
struct w_table *words;
    {
    int c;

    printf("\n");
    printf("Source =%s\n", s_file);
    printf("Distination =%s\n", d_file);
    printf("Word number = %d\n", words->word_no);
    for (c = 0; c <= words->word_no; c++)
        printf("  Word(%d)=|%s|\n", c, words->word[c]);
    }

/*  marking  */
mark(s_file, d_file, words)
char                *s_file,
                    *d_file;
struct w_table *words;
    {
    int  s_er,
         d_er,
         st,
         line_in();
    char  s_line[MAXLINE],
          d_line[MAXLINE];
    FILE *s_fp,
         *d_fp,
         *fopen();

    if ( (s_fp = fopen(s_file, "r")) == NULL )
        {
        printf("FILE(%s) can't open (for read)!\n",
                    s_file
               );
        exit(0);
        }
    if ( (d_fp = fopen(d_file, "w")) == NULL )
        {
        printf("FILE(%s) can't open (for write)!\n",
                    d_file
               );
        exit(0);
        }
    while (     ( (st = line_in(s_fp, s_line)) != EOF )
            && ( st != EOD )
```

```
        )
        {
        mark_buf(s_line, d_line, words);
        s_er = fputs(s_line, d_fp);
        d_er = fputs(d_line, d_fp);
        if ( (s_er != NOERR) || (d_er != NOERR) )
            {
            printf("FILE(%s) write error!\n", d_file);
            exit(0);
            }
        }

    fclose(s_fp);
    fclose(d_fp);
    }

/*  marking on buffer  */
mark_buf(s_line, d_line, words)
char              *s_line,
                  *d_line;
struct w_table *words;
    {
    char *line;
    int  n,
         ptr,
         search();

    for (n = 0; n < strlen(s_line) - 1; n++)
        d_line[n] = '_';
    d_line[n] = '\n';
    d_line[n + 1] = '\0';

    for (n = words->word_no; n >= 0; n--)
        {
        line = s_line;
        for (ptr = 0;line <= &s_line[MAXLINE];)
            {
            ptr = search(line, words->word[n]);
            line = &line[ptr];
            if (ptr <= MAXLINE)
                {
                d_line[line - s_line] = n + '0';
                line++;
                }
            else
                break;
            }
        }
    }

/*  one line input from source file  */
int line_in(fp, line)
FILE *fp;
char *line;
    {
    int c,
```

```
          ptr;

    ptr = 0;
    while (    ((c = getc(fp)) != EOF)
               && (c != EOD)
               && (c != '\n')
              )
          line[ptr++] = c;
    line[ptr - 1] = '\n';
    line[ptr] = '\0';
    return(c);
    }

/*  keyword search on buffer  */
int search(line, key_word)
char *line,
     *key_word;
    {
    int ptr,
        l_len,
        w_len;

    w_len = strlen(key_word);
    l_len = strlen(line);
    for (ptr = 0; ptr <= l_len; ptr++)
        {
        if ( strncmp(&line[ptr], key_word, w_len)
             == EQU
           )
             break;
        }
    if (ptr > l_len)
        return(MAXLINE + 1);
    else
        return(ptr);
    }
```

LIST FORMATTER

保存用リストの出力

■Program Outline

このLIST FORMATTERは，各種のプログラムの保存用ソースリストを，桁数や1ページあたりの行数を指定することにより，プリンタに出力します．各ページの先頭にはプログラムのタイトルとページ数を出力することもできるうえに，ラインNo.を付加することもできます．

■Explanation

COMファイル名はPLST.COMです．
コマンドラインからの入力は次のように行います．

A＞PLST 〔OPTION〕 FILENAME

オプションは表1のとおりです．P，N，SのオプションはP，N，Sのそれぞれを付加することによってONされます．それ以外のオプションは，たとえば1ページ行数を指定する場合は“L80”というように，オプションの後に指定する数値を付けて入力します．

例 A＞PLST NL60 TEST.C
（TEST.CをラインNo.を付けずに1ページ60行で出力）

表1

オプション	例	無指定時	機能
P	—	—	ページNo.の出力をしない
N	—	—	ラインNo.の出力をしない
T	T 4	8	タブをn文字に展開
M	M 5	0	レフトマージン
C	C 60	80	1行桁数
L	L 50	60	1ページ行数
S	—	—	画面に出力する

■Source List

```
/*                                    */
/*    List formatter  Rev 1.0         */
/*                                    */
/*      Copyright (C) 1984            */
/*     Program  by  M.Hanari          */
/*                                    */

#include "stdio.h"
#include "ctype.h"

#define ON  1
#define OFF 0

#define CPM_EOF 0x1a

/*  default parameter  */

int pg_nm = ON,         /*  page   number   */
    ln_nm = ON,         /*  line   number   */
    tab_m =  8,         /*  tab    margin   */
    lf_mg =  0,         /*  left   margin   */
    colum = 80,         /*  column length   */
    pg_ln = 60,         /*  page   length   */
    sc_md = OFF;        /*  screen output   */

char name[15];

FILE *fp,*lst;

main(argc,argv)
int argc;
char *argv[];
    {
    FILE *fopen();
    char dev[5];
    int i;

    if (argc != 1)
        {
        if (argc > 2)
            for (i = 1;i < argc - 1;i++)
                op_chk(argv[i]);
        strcpy(name,argv[argc - 1]);
        if ((fp = fopen(name,"r")) != NULL)
            {
            if (sc_md)
                strcpy(dev,"CON:");
            else
                strcpy(dev,"LST:");
            if ((lst = fopen(dev,"w")) != NULL)
                {
                lst_out();
                fclose(lst);
                }
```

```
                fclose(fp);
                }
            }
        }

/*  option check  */
op_chk(s)
char *s;
    {
    while (*s != '\0')
        switch(toupper(*s++))
            {
            case 'P':
                pg_nm = 1 - pg_nm;
                break;
            case 'N':
                ln_nm = 1 - ln_nm;
                break;
            case 'T':
                tab_m = get_num(&s);
                break;
            case 'M':
                lf_mg = get_num(&s);
                break;
            case 'C':
                colum = get_num(&s);
                break;
            case 'L':
                pg_ln = get_num(&s);
                break;
            case 'S':
                sc_md = ON;
                break;
            default:
                s++;
                break;
            }
    }

/*  get number  */
get_num(sp)
char **sp;
    {
    char n_buf[5];
    int i;

    for (i = 0;isdigit(**sp);i++)
        {
        n_buf[i] = **sp;
        ++(*sp);
        }
    n_buf[i] = '\0';
    return(atoi(n_buf));
    }
```

```
/*  list out  */
#define LIN_MAX 200

lst_out()
    {
    char l_buf[LIN_MAX];
    int  page = 0,
         line = 0,
         plin = 0,
         i;

    colum = colum - lf_mg - (ln_nm * 7);
    f_feed(++page);
    for (;;)
        {
        if (plin >= pg_ln)
            {
            f_feed(++page);
            plin = 0;
            }
        if ((i = getc(fp)) == EOF || i == CPM_EOF)
            break;
        l_mrg(++line);
        *l_buf = i;
        fgets(&l_buf[1],LIN_MAX,fp);
        plin += lin_out(l_buf);
        }
    }

/*  form feed & page print  */
f_feed(p)
int p;
    {
    int i;

    if (pg_nm)
        fprintf(lst,"\f  Title : %-14s%50s%03d\n\n"
                                        ,name,"Page : ",p);
    else
        fprintf(lst,"\f");
    }

/*  left margin & line number print  */
l_mrg(l)
int l;
    {
    int i;

    for (i = 0;i < lf_mg;i++)
        putc(' ',lst);
    if (ln_nm)
        fprintf(lst,"%5d: ",l);
    }

/*  1 line print  */
lin_out(buf)
char buf[];
```

```
    {
    int i = 0,
        t = 0,
        l = 1;

    char c;

    while((c = buf[i]) != '\n' && c != '\0')
        {
        if (buf[i] == '\t')
            {
            i++;
            putc(' ',lst);
            while (((i + t) % colum) % tab_m != 0)
                {
                putc(' ',lst);
                t++;
                if (! ((i + t) % colum))
                    {
                    lf_sp();
                    l++;
                    }
                }
            }
        else
            {
            putc(buf[i++],lst);
            if (! ((i + t) % colum))
                if ((c = buf[i]) != '\n' && c != '\0')
                    {
                    lf_sp();
                    l++;
                    }

            }
        }
    fprintf(lst,"\n");
    return(l);
    }

/*  cr & left space  */
lf_sp()
    {
    int i;

    fprintf(lst,"\n");
    for (i = 0;i < lf_mg + (ln_nm * 7);i++)
        putc(' ',lst);
    }
```

LOOK

ソースファイルの構造を把握するために

■Program Outline

C言語のソースファイルでは，中カッコ{ }は主に関数のボディーや構造体，あるいは制御構造などの始まりと終わりを表わすために用いられています．

このLOOKは，アスキーファイル中の多重の中カッコの深さを調べ，設定した深さよりもさらに深い中カッコに対しては，中カッコの外側だけをターミナルに表示します(ただし，どの深さでも中カッコ自身は表示します)．

たとえば，中カッコの深さを1に設定(オプションパラメータとして入力)した場合には，ソースファイル中で定義された関数のヘッダ部を見ることができます．

■Explanation

COMファイルはLOOK.COMとします．

コマンドラインからの入力は次のようになります．

LOOK FILENAME〔depth〕

FILENAMEは内容を見たいアスキーファイルとエクステンションです．

depthは多重の中カッコの深さを指定するためのオプションです．このdepthを指定しない場合は，0にセットされます．深さが0の場合あるいはマイナスの場合には，中カッコのみを表示します．また，ファイル内の中カッコの深さより指定を深くした場合には，ファイルをすべて表示します．

なおこのLOOKコマンドでは，ダブルコーテーションマーク”やシングルコーテーションマーク’で囲まれた内容や，コメントの内容としての中カッコは，中カッコとしては扱われません．

サンプルファイル

```
A>TYPE EXAMPLE2.C
/*----- example program source -----*/
main()
    {
    int c;

    for (c = 0; c < 10; c++)
        {
        printf("%d is ", c);
        if (c % 2)
            printf("{ODD}.\n");
        else
            printf("{EVEN}.\n");
        }
    }
```

出力例

```
A>LOOK EXAMPLE2.C
{{}}

A>LOOK EXAMPLE2.C 1
/*----- example program source -----*/
main()
    {{}}


A>LOOK EXAMPLE2.C 2
/*----- example program source -----*/
main()
    {
    int c;

    for (c = 0; c < 10; c++)
        {}
    }


A>LOOK EXAMPLE2.C 3
/*----- example program source -----*/
main()
    {
    int c;

    for (c = 0; c < 10; c++)
        {
        printf("%d is ", c);
        if (c % 2)
            printf("{ODD}.\n");
        else
            printf("{EVEN}.\n");
        }
    }
```

■Source List

```
/*                            */
/*  LOOK                      */
/*                            */
/*     Aug.1984               */
/*  Programed by M.Hanari     */
/*                            */

#include "b:stdio.h"
#define CPM_EOF 0x1a

main(argc, argv)
int argc;
char *argv[];
    {
    int level = 0, atoi();
    char *l_ptr;

    if ((argc == 2) || (argc == 3))
        {
        if (argc == 3)
            level = *argv[2] - '0';
        look(argv[1], level);
        }
    else
        {
        printf("Bad command line input!\n");
        printf("  Usr: LOOK filespec. [nesting level]\n");
        }
    }

look(f_name, level)
char *f_name;
int level;
    {
    int c;
    FILE *fp, *fopen();

    if ((fp = fopen(f_name, "r")) != 0)
        {
        while ( (c = getc(fp)) != EOF
                && c != CPM_EOF
              )
            putchar(convert(c, level));
        putchar('\n');
        }
    else
        puts("Bad file access!");
    }

#define OUT 0
#define SINGLE_Q 1
#define DOUBLE_Q 2
#define ON 1
#define OFF 0
```

```
convert(c, level)
char c;
int level;
    {
    static int depth = 0;
    int q;

    if ((q = q_check(c)) == OUT)
        {
        if (c == '{')
            {
            depth++;
            }
        else if (c == '}')
            {
            depth--;
            }
        }
    if (depth >= level)
        {
        if (q != OUT)
            c = '\0';
        else if ( !((c == '{') || (c == '}')))
            c = '\0';
        }
    return(c);
    }

q_check(c)
char c;
    {
    static int quot = OUT, flag_d = OFF, flag_s = OFF;

    if (quot == DOUBLE_Q)
        {
        if (flag_d == ON)
            flag_d = OFF;
        else
            {
            if (c != '\"')
                {
                if (c == '\\')
                    flag_d = ON;
                }
            else
                quot = OUT;
            }
        }
    else
        {
        if (quot == SINGLE_Q)
            {
            if (flag_s == ON)
                flag_s = OFF;
            else
                {
```

```
                if (c != '\'')
                    {
                    if (c == '\\')
                        flag_s = ON;
                    }
                else
                    quot = OUT;
                }
            }
        else
            {
            if (c == '\"')
                quot = DOUBLE_Q;
            else
                {
                if (c == '\'')
                    quot = SINGLE_Q;
                }
            }
        }

    return(quot);
    }
```

CALC

逆ポーランド表記による計算とスタックの表示

■Program Outline

このプログラムは算術式を逆ポーランド表記の式に変換して，計算を行います．また，算術式を逆ポーランド表記に変換する過程と計算する過程をトレース(画面に表示)します．トレースは，モードの設定により実行の有無を選択することもできます．このプログラムでは，扱える数は単精度の実数で指数表記は許されてはいません．

■Explanation

COMファイル名はCALC.COMです．次のように実行してください．

① コマンド名を入力します．

CALC

② トレースのモードを指定します．トレースのモードはAからDまでの4つあり，それぞれ次のようになっています．

A：逆ポーランド変換，計算ともにトレースせずにすぐに計算結果を表示する．

B：逆ポーランド変換のときのみトレースする．計算過程は表示せずに，すぐに計算結果を表示する．

C：逆ポーランド変換はトレースせず，計算過程のみトレースして結果を表示する．

D：逆ポーランド変換，計算ともにトレースして結果を表示する．

トレースの指定は，英字(AからDまでのうち1文字で大文字でも小文字でもよい)を入力し，キャリッジリターンを入力します．

現在設定されているモードは，画面にプロンプトして表示されます．たとえば，カーソルの左に次のようなプロンプトが表示されている場合には，

```
A ]
```

トレースのモードはAということになります．もし，モードを変更をする必要がない場合には，次の③へ進みます．

③ 計算(あるいはトレース)する算術式を入力します．入力する算術式に許される文字，記号は数字の0から9まで，小数点，負符号，＋，－，*，／および小カッコ()だけです．等号＝は許されません．また変数も許されません．入力する算術式の長さは199文字(MAX_BUF-1)まで許されますが，あまり長いとトレースしたときにコンソールの1画面に表示しきれず見ずらくなります．

カッコは入力文字数が許す限り多重で用いることができますが，左カッコと右カッコの対応はチェックしていないので，正しく対応していない場合の動作は保障できません．

キャリッジリターンの入力によって算術式の入力が終了して，次の④の逆ポーランド変換が始まります．

④ 逆ポーランド変換の実行です．逆ポーランド変換は入力された算術式を左から右に評価していきます．逆ポーランド変換をトレースする（トレースモードがBまたはDのとき）場合にはこのようすが表示されます．トレースは，1ステップ終わるたびに入力待ちとなり，キャリッジリターンが入力されると次のステップに移ります．

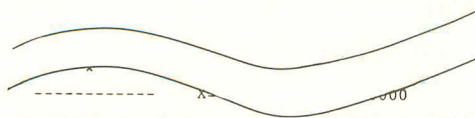
⑤ 計算の実行です．逆ポーランド変換が終了すると，すぐに計算を始めます．計算は逆ポーランド変換でトレースしたときに，画面に表示されている左のスタックに積まれた逆ポーランド表記の式を，上から順に取り出して計算していき，スタックが空になると終了します．計算をトレースする（トレースモードがCまたはDのとき）場合には，このようすが表示されます．

④と同様にトレースは1ステップずつ進みます．

⑥ 結果を表示します．トレースのモードに関係なく，結果を表示して，②の入力待ち状態になります．トレースモードを変える場合は，②のトレースモード変更の入力をし，変えない場合には次に計算(あるいはトレース)したい算術式を③に従って入力します．

⑦ CALCコマンドを終了させてOSに戻るときは，コントロールCの入力によってブレークします．

出力例

```
                /// Reversed Polish Calculation ///

                     A : Normal mode
                     B : Convert trace only
                     C : Caluclation trace only
                     D : Full trace

A]b
B]1.2+3
1.2+3
  ^

|     1.2 |    |         |
 ----------     ---------

/// Answer ///

|       + |    |         |
|       3 |    |         |
|     1.2 |    |         |
 ----------     ---------

1.2+3  ==>  1.2 3 +

] ANSWER=      4.2000

B]c
C]1.2+3
1.2+3  ==>  1.2 3 +

>> PUSH DATA <<

1.2 3 +
^
|   1.2000 |
 ------------    X=0.000000 Y=0.000000

>> PUSH (X + Y) <<

1.2 3 +
      ^
|   4.2000 |
 ------------    X=1.200000 Y=3.000000

>> POP ANSWER <<

1.2 3 +
        ^
 ------------    X=1.200000 Y=3.000000

] ANSWER=      4.2000

C]a
A]1.23+4.56*2.5
1.23+4.56*2.5  ==>  1.23 4.56 2.5 * +

] ANSWER=    12.6300
```

■Source List

```
/*                                                   */
/*   Reversed Polish Calculation With Tracer         */
/*                                                   */
/*        Aug.1984   Gijutsuhyouronsha               */
/*    Programed  by  M.Hanari & M.Matsuzaki          */
/*                                                   */

#include "b:ctype.h"

#define MAX_BUF 200

#define EOF 1
#define EMK '\0'

#define TRUE  1
#define FALSE 0

char a[MAX_BUF];
char buf[MAX_BUF];
char tm;

main()
    {
    int ct;

    title();
    for (;;)
        {
        while (key_in() != TRUE)
            ;
        putchar('\0');
        chenge();
        printf("%s  ==>  ",a);
        prts();
        calc();
        for (ct = 0;ct < 100;ct++)      /*  buffer clear  */
            buf[ct] = '\0';
        }
    }

/*  title & menu  */
title()
    {
    puts("\t///  Reversed Polish Calculation  ///\n");
    puts("\t     A : Normal mode");
    puts("\t     B : Convert trace only");
    puts("\t     C : Caluclation trace only");
    puts("\t     D : Full trace\n");
    tm = 'A';
    }

/*  get formula from console  */
key_in()
    {
```

```
    char c;

    printf("%c]",tm);
    gets(a);
    if (((c = *a) >= 'A' && c <= 'D') ||
              ( c >= 'a' && c <= 'd')     )
        tm = (islower(*a) ? toupper(*a): *a);
    else
        switch(*a)
            {
            case '\0':
                break;
            default:
                return(TRUE);
            }
    return(FALSE);
    }

/*  Convert to reversed Polish style   */
char fbuf[MAX_BUF];
char *ptr,*bufptr,*fbufptr;

chenge()
    {
    ptr = a;
    bufptr = buf;
    fbufptr = fbuf;
    while (*ptr != '\0')
        {
        if (number(ptr) == TRUE)
            mov_numb();
        else
            if (operator() == FALSE)
                continue;
        trace();
        }
    if (fbufptr > fbuf)
        while (fbufptr > fbuf)
            mov_sub();
    *bufptr = EOF;
    answer();
    }

/*  operators' process  */
operator()
    {
    char c;

    switch (*ptr)
        {
        case '(':
            *fbufptr++ = *ptr++;
            break;
        case ')':
            while (fbufptr[-1] != '(')
                mov_sub();
```

```
                fbufptr--;
                ptr++;
                break;
            case '*':
            case '/':
                if (fbufptr > fbuf)
                    if ((c = fbufptr[-1]) == '*' || c == '/')
                        mov_sub();
                *fbufptr++ = *ptr++;
                break;
            case '+':
                mov_op();
                break;
            case '-':
                if ( ptr > a &&
                    (number(&ptr[-1]) == TRUE || ptr[-1] == ')') )
                    mov_op();
                else
                    {
                    *bufptr++ = *ptr++;
                    return(FALSE);
                    }
                break;
            default:
                ptr++;
                break;
            }
        return(TRUE);
        }

/*  trace-mode check and print   */
trace()
        {
        char *ct;

        if (tm == 'B' || tm == 'D')
            {
            puts(a);
            for (ct = a ;++ct < ptr; putchar(' '))
                ;
            puts("^\n");
            stc_prt(bufptr,fbufptr);
            getchar();
            }
        }

/*  print reversed Polish style   */
answer()
        {
        if (tm == 'B' || tm == 'D')
            {
            puts("///  Answer  ///\n");
            stc_prt(bufptr,fbufptr);
            }
        }
```

```
/*  stack print on trace-mode   */
stc_prt(bufpt1,bufpt2)
char *bufpt1,*bufpt2;
    {
    int dps1;

    bufpt1--;
    bufpt2--;
    dps1 = dps_cnt(bufpt1);
    while ((dps1 >= 0) || (bufpt2 >= fbuf))
        {
        if (dps1 == (bufpt2 - fbuf))
            {
            printf("| %8s |     |     %c     |\n"
                  ,bufpt1 = back_ptr(bufpt1) ,*bufpt2--);
            bufpt1--;
            dps1--;
            }
        else
            if (dps1 > (bufpt2 - fbuf))
                {
                printf("| %8s |     |           |\n"
                      ,bufpt1 = back_ptr(bufpt1));
                bufpt1--;
                dps1--;
                }
            else
                printf("|            |     |     %c     |\n",*bufpt2--);
        }
    puts(" ----------        ---------\n");
    }

/*  number check   */
number(n_ptr)
char *n_ptr;
    {
    char c;

    if (((c = *n_ptr) >= '0' && c <= '9') || c == '.')
        return(TRUE);
    return(FALSE);
    }

/*  move number   */
mov_numb()
    {
    char *c;

    do
        {
        *bufptr++ = *ptr++;
        c = bufptr;
        }
    while (nul_in() == c);
    }
```

```
/*  endmark set of numbers' end  */
nul_in()
    {
    if (number(ptr) == FALSE)
        *bufptr++ = EMK;
    return(bufptr);
    }

/*  move operator  */
mov_op()
    {
    if (fbufptr > fbuf)
        if (fbufptr[-1] != '(')
            mov_sub();
    *fbufptr++ = *ptr++;
    }

/*  move operator sub  */
mov_sub()
    {
    *bufptr++ = *--fbufptr;
    *bufptr++ = EMK;
    }

/*  count depth of bufer */
dps_cnt(tp_ptr)
char *tp_ptr;
    {
    int dps = 0;

    for (tp_ptr--;buf <= tp_ptr ;tp_ptr--)
        {
        if (*tp_ptr == EMK)
            dps++;
        }
    return(dps);
    }

/*  pointer back for stack print  */
back_ptr(tp_ptr)
char *tp_ptr;
    {
    while (*--tp_ptr != EMK)
        if (buf == tp_ptr)
            return(buf);
    return(++tp_ptr);
    }

/*  print reversed Polish style  */
prts()
    {
    char *p_ptr;

    for (p_ptr = buf; *p_ptr != EOF ;p_ptr++
        {
        if (*p_ptr == EMK)
```

```
                putchar(' ');
            else
                putchar(*p_ptr);
            }
        puts("\n");
        }

/*  Calculation block */
#define EOW 0
#define EOF 1
#define VAL 2
#define OP 3

#define MAX_STK 100

double stack[MAX_STK], *stack_p;

/*  Revrsed Polish formula calculation  */
calc()
    {
    char c, *word_p;
    int atr();
    double ope(), pop(), atof(), x = 0, y =0, z, answer;

        stack_p = stack;

    word_p = buf;
    while (atr(word_p) != EOF)
        {
        if (atr(word_p) == VAL)
            {
            push(atof(word_p));
            push_w(word_p, x, y);
            }
        else
            {
            y = pop();
            pop_y(word_p, x, y);

            x = pop();
            pop_x(word_p, x, y);

            z = ope(word_p, y, x);

            push(z);
            push_z(word_p, x, y, z);
            }
        next(&word_p);
        }
    answer = pop();
    if ((tm == 'C') || (tm == 'D'))
        {
        printf(">> POP ANSWER <<\n\n");
        dsp_stk(word_p, x, y);
        }
```

```
    printf("] ANSWER= %10.4f\n", answer);
    printf("~~~~~~~~~~~~~~~~~~~~~\n");
    }

/*  word attribute check  */
int atr(word_p)
char *word_p;
    {
    if (*word_p == EOF)
        return(EOF);
    else if ((*word_p == '+') ||
             (*word_p == '-') ||
             (*word_p == '*') ||
             (*word_p == '/')   )
        return(OP);
    else
        return(VAL);
    }

/*  word push  */
push_w(word_p, x, y)
char *word_p;
double x, y;
    {
    if ((tm == 'C') || (tm == 'D'))
        {
        printf(">> PUSH DATA <<\n\n");
        dsp_stk(word_p, x, y);
        while (getchar() != '\n')
            ;
        }
    }

/*  Y pop  */
pop_y(word_p, x, y)
char *word_p;
double x, y;
    {
    if ((tm == 'C') || (tm == 'D'))
        {
        printf(">> POP Y <<\n\n");
        dsp_stk(word_p, x, y);
        while (getchar() != '\n')
            ;
        }
    }

/*  X pop  */
pop_x(word_p, x, y)
char *word_p;
double x, y;
    {
    if ((tm == 'C') || (tm == 'D'))
        {
        printf(">> POP X <<\n\n");
        dsp_stk(word_p, x, y);
        while (getchar() != '\n')
```

```
            ;
        }
    }

/*  results of operate push */
push_z(word_p, x, y, z)
char *word_p;
double x, y, z;
    {
    if ((tm == 'C') || (tm == 'D'))
    {
    printf(">> PUSH (X %c Y) <<\n\n", *word_p);
    dsp_stk(word_p, x, y);
    while (getchar() != '\n')
        ;
    }
}

/*  value data push  */
push(f_data)
double f_data;

    {
    *stack_p++ = f_data;
    }

/*  value data pop  */
double pop()
    {
    return(*--stack_p);
    }

/*  operation on operator  */
double ope(op_ptr, val_1, val_2)
char *op_ptr;
double val_1, val_2;
    {
    char c;
    double ans;

    switch (*op_ptr)
        {
        case '+':
            ans = val_2 + val_1;
            break;
        case '-':
            ans = val_2 - val_1;
            break;
        case '*':
            ans = val_2 * val_1;
            break;
        case '/':
            ans = val_2 / val_1;
            break;
        }
    return(ans);
    }
```

```
/*  word pointer set to next word   */
next(wp_ptr)
char **wp_ptr;
    {
    while (**wp_ptr != EOW)
    (*wp_ptr)++;
    (*wp_ptr)++;
    }

/*  display value stack top  */
dsp_stk(word_p, x, y)
char *word_p;
double x, y;
    {
    double *sp;

    dsp_ptr(word_p);
    for (sp = &stack_p[-1]; sp >= stack; sp--)
        printf("| %10.4f |\n", *sp);
    printf(" ------------     X=%f Y=%f\n", x, y);
    printf("\n\n");
    }

/*  display value stack point   */
dsp_ptr(word_p)
char *word_p;
    {
    char *buf_p;

    for (buf_p = buf; *buf_p != EOF; buf_p++)
        if (*buf_p == '\0')
            putchar(' ');
        else
            putchar(*buf_p);
    putchar('\n');

    for (buf_p = buf; buf_p != word_p; buf_p++)
        putchar(' ');
    printf("^\n");
    }
```

cat

ファイルとファイルを連結

■Program Outline

catは同名のUNIXの基本コマンドをMS-DOS V.2.11に移植したものです．本来の機能はファイルとファイルを連結(concatenate)することですが，UNIX上ではファイルの内容を出力させるのによく使われます．

入出力は標準入出力になっています．

■Explanation

EXEファイル名はcat.EXEとします．

コマンドラインの一般的な書式は次のようになります．

A＞cat　オプション　ファイル名1〔ファイル名2………〕

ファイルとファイルを連結するときは，書式のようにファイル名1，ファイル名2，………，というようにファイル名を入力して下さい．オプションはバークレー版のUNIX4.1BSDのものに準じています(表1)．

catの最も単純な利用例は，次のようにファイル名を一つ指定して，そのファイルを表示させることです．

A＞cat　test 1．c

出力例

```
A>cat test1.c
/* test program no.1 */
main(argc, argv)
int argc;
char *argv[];
{
        char esc = 27;

        while (--argc > 0)
                printf("%c%s", esc, *++argv);
}
```

表1

オプション	機　　　　能
－n	表示するファイルの先頭から順に行番号をつける
－nb	－nの機能に加え，空白行への行番号をはずす
－v	通常はマスクされるコントロールコードを表示する

また，2つ以上のファイルを指定して，続けて表示させることもできます．

A＞cat　test 1．c　test 2．c

ちょっとひねった使い方としては，小さなファイルを作成するのにも使うことができます．標準出力をリダイレクトしてファイル名とする方法です．実例をあげると，

A＞cat　＞sample

とすると，標準入力できるキーボードからの入力がそのままsampleという名のファイルになります．入力のを終えるときは，MS-DOSのファイル終了信号^Zを入力して，キャリジリターンを押して下さい(UNIXでは^D)．

この方法を使えば，コントロールコードの入ったファイルも作ることができます．

出力例
```
A>cat >sample
abcdefghijklmnopqrstuvwxyz
^G^D^D^T^D^T^Y  ^U      a^A
^Z
```

このようなファイルは通常は中身が表示されませんが,－vオプションを使用することにより内容を見ることができます．

出力例
```
A>cat sample
abcdefghijklmnopqrstuvwxyz
          a

A>type sample
abcdefghijklmnopqrstuvwxyz
          a

A>cat -v sample
abcdefghijklmnopqrstuvwxyz
^G^D^D^T^D^T^Y  ^U      a^A
```

■Source List

```
/*                                  */
/*    concatenate                   */
/*                                  */
/*    copyright (C) 1984            */
/*    programed by y.aihara         */
/*                                  */

#include <stdio.h>

struct {
     unsigned visual : 1,    ―――optsという構造体の定義各変数をbit単位で定義
          number : 1,
          blank : 1;
} opts;

main(argc, argv)
int argc;
char *argv[];
{
     FILE *fp, *fopen();
     char *s;

     while (--argc > 0 && *argv[1] == '-') ―――オプションチェック
          for (s = *++argv+1; *s; s++)
               switch (*s) {
               case 'v':
                    opts.visual = 1;  ―――vオプション
                    break;
               case 'n':
                    opts.number = 1;  ―――nオプション
                    break;
               case 'b':
                    opts.blank = 1;  ―――bオプション
                    break;
               default:
                    error("Usage: cat -vnb file....", NULL);
                                                        エラー
               }
     if (argc == 0)
          filecopy(stdin); ―――stdinファイルをコンソール画面に出力
     else
          while (argc-- > 0) {
               if ((fp = fopen(*++argv, "r")) == NULL) リードファイルオープン
                    error("cat: can't open %s", *argv); エラー表示
               filecopy(fp); ―――ファイルをコンソール画面に出力
               fclose(fp); ―――ファイルのクローズ
          }
}

#define MAXLEN 256

filecopy(fp)
FILE *fp;
{
```

```
    char line[MAXLEN];
    int lineno = 0;

    while (fgets(line, MAXLEN, fp) != NULL) {
        if (opts.number && (!opts.blank || *line != '\n'))

            printf("%3d  ", ++lineno); ――――ラインナンバーの表示
        if (opts.visual)
            look(line); ――――Vフラグ ON なら LOOK を実行
        else
            printf("%s", line); ――――ファイル本体の表示

    }
}

look(s)
char *s;
{

    for ( ; *s; s++) ――――コントロールコードのチェック
        if (iscntrl(*s))
            switch (*s) {
            case '\n': ――――ラインフィードコードか？
            case '\t': ――――タブコードか？
            case '\b': ――――バックスペースコードか？
            case '\r': ――――リターンコードか？
            case '\f': ――――フォームフィードか？
                putchar(*s); ――――以上ならそのまま出力
                break;
            case 0177: ――――デリートコードか
                printf("^?");
                break;
            default:                        上以外のコントロール
                printf("^%c", *s | 0100);――コードなら“^”をつ
                break;                                けて表示
            }
        else
            putchar(*s);――――コントロールコード以外ならそのまま表示
}
```

エラーモジュール

```
/* error masage print and exit */

#include <stdio.h>

error(s1, s2)
char *s1, *s2;
{
        fprintf(stderr, s1, s2);
        fprintf(stderr,"\n");
        exit(1);
}
```

head

プログラムのヘッダを見る

■Program Outline

headはUNIXのバークレー版にある同名のコマンドをMS-DOS V.2.11に移植したものです．

基本的な機能としては，ファイルの最初の部分を先頭から指定した行数だけ出力することです．これを利用して，各プログラムのヘッダの部分のみを見ることに使います．また，標準入出力を使っているので，コマンドのパイプラインのフィルタ・コマンドとしても利用できます．

■Explanation

EXEファイル名はhead.EXEとします．

単独で使うときのコマンドラインからの入力は，次のように行います．

A＞head　－行数　ファイル名１〔ファイル名２………〕

この場合，行数を省略すると，10を指定したことになります．ファイル名はいくつ続けてもかまいません．オプションはありません．

最も単純な使い方は次のようになります．

```
A>head test1.c
/* test program no.1 */
main(argc, argv)
int argc;
char *argv[];
{
        char esc = 27;

        while (--argc > 0)
                printf("%c%s", esc, *++argv);
}
```

このように，Cのソースファイルtest１.cの最初から10行目までを表示されます．また，行数を指定することにより，表示する範囲を変えることができま

す．ファイル名はいくつでも並記できますので，複数のファイルの初期設定の比較などに便利です．

出力例
```
A>head -5 test1.c
/* test program no.1 */
main(argc, argv)
int argc;
char *argv[];
{
```

このプログラムは標準入出力によって入出力を行っていますので，パイプラインの中でフィルタ・コマンドとして利用することもできます．

具体的な使用例をあげると，数値の入ったデータファイルの中のデータを小さい順に3つ表示する仕事は，次のようにコマンドを組み合わせることによって実現できます．

出力例
```
A>type sample10
 452
1124
  45
 487
1049
  10
2045
2199
  25

A>type sample10|sort|head -3
  10
  25
  45
```

■Attention

リンク時にcatに掲載されているエラーモジュールをリンクしてください．

エラーメッセージ

Cソースの移植

OPTIMIZING Cで書いてあるプログラムを他のCに移植するときは次の点に留意してください．

- 文字型に関する関数（isalpha, isdigit, tolower, etc.）を使用しているプログラムでは，"ctype. h" ファイルをincludeする．
- gets関数の書式が他のCと異なっているので，移植するCの書式に変える．
- ビット・フィールドをサポートしていないCは，マクロ定義とビット演算を使う（K＆Rの『プログラミング言語C』を参照されたし）．

■Source List

```
/*                              */
/*    head                      */
/*                              */
/*    copyright (C) 1984        */
/*    programed by y.aihara     */
/*                              */

#include <stdio.h> ──────標準入出力ヘッダを含む

main(argc, argv)
int argc;
char *argv[];
{
     FILE *fp, *fopen();
     int length = 10; ──────行数
     int title = 0; ──────タイトルを表示するかのフラグ

     if (argc > 1 && *argv[1] == '-') {  ┐
          --argc;                        ├──────オプションの行数の指定
          length = atoi(*++argv+1);      ┘
     }
     if (argc == 1)
          filecopy(stdin, length); ────ファイルの指定がなければ標準入力
     else {                              から読み込む
          if (argc > 2)  ┐
               title = 1;┘──────ファイルが複数ならタイトル表示フラグを ON
          while (--argc > 0) {
               if ((fp = fopen(*++argv, "r")) == NULL)─ファイルを
                    error("head: can't open %s", *argv); オープン
               if (title)                               ┐フラグがONな
                    printf("==> %s <==\n", *argv);      ┘らタイトル表示
               filecopy(fp, length);─指定ファイルから読んで指定行数表示
               fclose(fp);
               if (argc > 1)          ┐ ────次のファイルがあれば空白行を出力
                    putchar('\n');    ┘
          }
     }
}

#define MAXLEN 256 ──────1行の最大文字数

filecopy(fp, len) ──────ファイルの先頭を表示
FILE *fp;
int len;
{
     char line[MAXLEN]; ──────1行入力用のバッファ

     while (len-- > 0 && fgets(line, MAXLEN, fp)) ┐ 指定行数を超
          printf("%s", line);                     │ えるかEOFに
}                                                 ┘ なるまで出力
```

uniq

ファイルの要素の抽出法と要素数を知る方法

■Program Outline

このプログラムは，UNIX上の同名のコマンドをMS-DOS V.2.11上に実現したものです．基本的な機能は，隣り合った行の内容を比較し，重複している場合はその行を削除することです．実際は，sortと組み合わせてファイルの要素を抽出したり，要素の数を知るために使われます．

これ以外に，パイプライン上のフィルタコマンドとしても使用できます．

■Explanation

EXEファイル名はuniq.EXEとします．

コマンドラインからの入力は次のような書式になります．

A＞uniq 〔オプション〕 ファイル名1 〔ファイル名2〕

オプションの機能は表1を参照して下さい．ファイル名1に入力ファイル名，ファイル名2に出力ファイル名を指定します．

たとえば，sampleというファイルの中の重複しているデータを削除して，sample. newというファイルに出力すると，次のようになります．

```
A>cat sample
108
125
125
132
148
148

A>uniq sample sample.new

A>cat sample.new
108
125
132
148
```

ただし，uniqは隣り合った行だけを比較するため，次に掲げるsampleのようなファイルについては効果を発揮しません．

```
A>cat sample
125
148
132
108
125
148
```

このような場合には，パイプラインを利用してsortコマンドと組み合わせて使います．

```
A>cat sample | sort | uniq
108
125
132
148
```

パイプライン上では標準出力へ出力することになるため，ファイルを作るときはリダイレクト機能を使ってやる必要があります．

オプション	機　　能
－c	各行の先頭に重複回数を表示する
－u	重複しない行のみを表示する
－d	重複した行のみを表示する

エラーメッセージ

Cのデバッグ

Cプログラミングで間違いやすいミスをいくつかあげます．デバッグの参考にしてください．

- 不注意による場合 …… スペルミスや'；'および'，"の付け忘れや'(' および'{'の対応が悪いなど．
- 制御の流れ……ifとelseのつづり，ループ本体の範囲（'｛'の付け忘れ)，case文中のbreakの有無などが違っている場合．
- 式の評価順序……カッコでくくれば安心です．
- 境界条件……ループの脱出条件，型の精度および桁あふれ，文字列操作したときのヌルキャラの付け忘れなど，主に無限ループを引き起こすバグ．
- その他……関数の引数の型が一致していないポインタ型と整数型の違法な演算による場合など．

■Source List

```
/*                                   */
/*     uniq                          */
/*                                   */
/*     copyright (C) 1984            */
/*     programed by y.aihara         */
/*                                   */

#include <stdio.h> ──────標準入出力ヘッダをインクルード
#define MAXLEN 256

struct {
     unsigned uniq : 1,  ┐
          dup : 1,       ├──────構造体として uniq, dup, count を1bit で定義
          count : 1;     │
} opts;                  ┘

main(argc, argv)
int argc;
char *argv[];
{
     FILE *fp1, *fp2, *fopen();
     char *s;

     while (--argc > 0 && *argv[1] == '-')───オプションがあるかどうか
          for (s = *++argv+1; *s; s++)
               switch (*s) {            ┐
               case 'u':                │
                    opts.uniq = 1;      │
                    break;              │
               case 'd':                ├───各オプションに応じたフラグの設定
                    opts.dup = 1;       │
                    break;              │
               case 'c':                │
                    opts.count = 1;     │
                    break;              ┘
               default:
                    error("uniq: illegal option %c", *s);オプションが正常でないならエラーを表示
               }
     if (argc == 0)
          uniq(stdin, stdout);
     else if((fp1 = fopen(*++argv, "r")) == NULL)────リードファイルのオープン
          error("uniq: can't open %s", *argv);
     else if (argc == 1)       ┐
          uniq(fp1, stdout);   ┘────そのファイル名で uniq ルーチンへ
     else if ((fp2 = fopen(*++argv, "w")) == NULL)┐
          error("uniq: can't create %s", *argv);   ┘───ライトファイルのオープン
     else
          uniq(fp1, fp2); ──────リードファイル, ライトファイル設定により uniq ルーチンへ
}

uniq(fp1, fp2)
FILE *fp1, *fp2;
{
     char line[MAXLEN], buf[MAXLEN];
     int c = 0;
```

```
    *buf = '\0'; ―――――――buf の先頭にヌルコードを代入
    while (fgets(line, MAXLEN, fp1))―――― Line にファイル内容をリード
        if (strcmp(line, buf)) {
            putbuf(buf, c, fp2); ―――――――直前の行との比較
            c = 1;
            strcpy(buf, line);――バッファに現在のラインの内容をコピー
        } else
            c++;
    putbuf(buf, c, fp2);
}

putbuf(buf, c, fp)
char *buf; ―――――――バッファの内容と d の内容を各オプション設定に応じて表示する
int c;
FILE *fp;
{
    if (!opts.uniq && !opts.dup ||\
    opts.uniq && c == 1 ||\
    opts.dup && c > 1) {
        if (opts.count && c)
            fprintf(fp, "%4d ", c);
        fputs(buf, fp);
    }
}
```

tr

UNIXコマンドをMS-DOS上で実現

■Program Outline

このプログラムはUNIXのtrコマンドの機能をMS-DOS V. 2.11上で実現したものです．もちろん，標準入出力もサポートしています．

コマンドの名称はtranslate（変換）の略記です．その名の通り，数字を含む文字および文字式を，指定したものへ置き換えるはたらきをします．本来，UNIX上ではASCII文字だけをサポートしていましたが，このプログラムではカタカナもサポートしています．

■Explanation

EXEファイル名はtr.EXEとします．

コマンドの書式は，以下のようになります．

A＞tr　オプション　文字列1　文字列2

これにより文字列1で指定された文字列を，文字列2に変換します．文字列1と文字列2は，初めから順に対応していきます．このとき，文字列はひとかたまりで変換されるのではなく，1つ1つ変換されます．

出力例
```
A>tr abc ABC
abcdefg
ABCdefg
^Z
```

この場合，標準入力のキーボードから入力していますので，入力の終了時にはMS-DOSのファイル終了記号^Z（コントロールZ)を入力する必要があります．

また文字列はASCII順であれば，“－”を使って中間を省略することができます．次に示すサンプルファイルとそれを“＞”を使ってリダイレクトし，入力ファイルとした際の変換後の結果を見て下さい．

サンプルデータ

```
A>cat sample2
abcdefghijklmnopqrstuvwxyz
1234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
123456789:;<=>?@ABCDEFGHIJKLMN
```

変換後の結果

```
A>tr a-z A-Z <sample2
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
123456789:;<=>?@ABCDEFGHIJKLMN
```

また，－cオプションを利用すると，文字列1の指定を短くすませることができます．

出力例

```
A>cat sample20
1234567890

A>tr -c 123 a <sample20
123aaaaaaa
```

文字列1と文字列2の長さが異なっている場合は，どちらが長いかによって結果が変わってきます．

文字列1が長い場合には，文字列2の余りは文字列1の最後の文字に割り当てられます．逆の場合には，文字列2の余りは無視されます．

文字列中には特殊記号も自由に指定できます．

出力例(特殊記号)

```
A>tr a-z @ <sample2
@@@@@@@@@@@@@@@@@@@@@@@@@@
1234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
123456789:;<=>?@ABCDEFGHIJKLM
```

－dオプションを使うときには文字列2の指定は必要ありませんし，指定しても無視されます(表1)．

表1

オプション	機　　能
－d	文字列1の文字を削除
－c	文字列1以外の文字を指定
－s	置き換えた文字が連続する場合，1文字だけに圧縮する

■ Attention

リンク時にcatに掲載されているエラーモジュールをリンクしてください．

■Source List

```
/*                          */
/*    translate             */
/*                          */
/*    copyright (C) 1984    */
/*    program by y.aihara   */
/*                          */

#include <stdio.h>
#define MAXLEN 256
#define iskanji(x) x&0200

struct {
	unsigned delete : 1,
		squeeze : 1,
		complement : 1;
} opts;

main(argc, argv)
int argc;
char *argv[];
{
	char from[MAXLEN], to[MAXLEN], *s;
	char *index();
	int c, cc, i;

	while (--argc > 0 && *argv[1] == '-')
		for (s = *++argv+1; *s; s++)
		switch (*s) {
		case 'd':
			opts.delete = 1;
			break;
		case 's':
			opts.squeeze = 1;
			break;
		case 'c':
			opts.complement = 1;
			break;
		default:
			argc = 0;
			break;
		}
	if (argc != 1 && argc != 2)
		error("Usage: tr [opt] str1 str2", NULL);
	expand(*++argv, from);
	if (opts.complement)
		invert(from);
	if (argc == 1)
		strcpy(to, from);
	else {
		expand(*++argv, to);
		stretch(to, strlen(from));
	}
	cc = 0;
	while ((c = getchar()) != EOF)
```

```
          if (iskanji(c)) {
               printf("%c%c", c, getchar());
               continue;
          } else if (!(s = index(from, c)))
               putchar(cc = c);
          else if (opts.delete)
               continue;
          else if (!opts.squeeze || *(to + (s-from)) != cc)
               putchar(cc = *(to + (s - from)));
}

/* expand */

expand(s, t)
char *s, *t;
{
     char c;

     while (*s) {
          *t++ = *s++;
          if (*s == '-' && *(s-1) <= *(s+1)) {
               c = *(s-1)+1;
               while (c <= *(s+1))
                    *t++ = c++;
               s += 2;
          }
     }
     *t = '\0';
}

/* invert */

invert(s)
char *s;
{
     char buf[256], *t = buf;
     int c = 1;

     while (c < 0400)
          *t++ = c++;
     *t = '\0';
     squeeze(buf, s);
     strcpy(s, buf);
}

/* squeeze */

squeeze(s, t)
char *s, *t;
{
     char *p, *q;

     for (p = s; *p; p++) {
          for (q = t; *q; q++)
               if (*p == *q)
                    break;
```

```
            if (!*q)
                *s++ = *p;
        }
        *s = '\0';
}

/* stretch */

stretch(s, n)
char *s;
int n;
{
        int i = strlen(s);
        char c = s[i-1];

        while (i < n)
                s[i++] = c;
        s[i] = '\0';
}
```

od

ファイル内容を出力する豊富な機能

■Program Outline

odはUNIXの同名のコマンドをMS-DOS V.2.11上で実現したものです．これは，ファイルの内容をみる専用の命令ですから，catに比べると機能が豊富になっています．特に，catの－vオプションでは表示されない，タブやヌルなどの特殊記号も表示することができます．

MS-DOSのDUMPコマンドと併用すると有効です．

■Explanation

EXEファイル名はod.EXEとします．

コマンドラインでの入力は次のようになります．

A＞od 〔オプション〕ファイル名

入力するときに，オプションを省略した場合は，デフォルトは－oにセットされます．それぞれのオプションの機能については，表1を参照して下さい．この入力においては，ファイル名の指定は1つだけになっています．実行例を次に示します．

```
A>od sample1
0000000 131261 132263 133665 134266 135271 136273 137275 140
277
0000020 142701 142303 143305 144307 145311
```

－cオプションを指定した場合，通常は表示されない文字が表2に示すように変換された状態で出力されます．

```
A>od -c sample1
0000000   1   2   3   4   5   7   6   8   9   :   ;   <   =
  >   ?   @
0000020   A   E   C   D   E   F   G   H   I   J   \n
```

－b，－d，－xの各オプションに関しては，次の実行例を参照して下さい．

```
A>od -b sample1
0000000  261  262  263  264  265  267  266  270  271  272  2
73  274  275  276  277  300
0000020  301  305  303  304  305  306  307  310  311  312  0
12

A>od -d sample1
0000000 -19791 -19277 -18507 -18250 -17735 -17221 -16707 -16
193
0000020 -14911 -15165 -14651 -14137 -13623

A>od -x sample1
0000000 B2B1 B4B3 B7B5 B8B6 BAB9 BCBB BEBD C0BF
0000020 C5C1 C4C3 C6C5 C8C7 CAC9
```

表1

オプション	表示単位	表示形式
－o	16bit （1word）	8進数
－c	8 bit （1byte）	ASCIIコード
－b	8 bit （1byte）	8進数
－d	16bit （1word）	10進数
－x	16bit （1word）	16進数

表2

機能名	JISコード（&H）	表示
null（ヌル）	00	／0
back space（バックスペース）	08	／b
formfeed（書式送り）	0C	／f
newline（改行）	0A	／n
carriadge return（復帰）	0D	／r
tab（水平タブ）	09	／t
その他	—	3桁の8進数

■Attention

リンク時にcatに掲載されているエラーモジュールをリンクしてください．

■Source List

```
/*                                */
/*    octal dump                  */
/*                                */
/*    copyright (C) 1984          */
/*    programed by y.aihara       */
/*                                */

#include <stdio.h>

#define ASCII   001
#define OCTAL   002
#define BYTE    004
#define DECIMAL 010
#define HEXDECI 020

main(argc, argv)
int argc;
char *argv[];
{
	FILE *fp, *fopen();
	char *s;
	int mode = OCTAL;

	while (--argc > 0 && *argv[1] == '-')
		for (s = *++argv+1; *s; s++)
			switch (*s) {
			case 'c':
				mode = ASCII;
				break;
			case 'o':
				mode = OCTAL;
				break;
			case 'b':
				mode = BYTE;
				break;
			case 'd':
				mode = DECIMAL;
				break;
			case 'x':
				mode = HEXDECI;
				break;
			default:
				argc = 0;
				break;
			}
	if (argc != 1)
		error("Usage: od -cobdx file", NULL);
	if ((fp = fopen(*++argv, "r")) == NULL)
		error("od: can't open %s", *argv);
	if (mode == ASCII || mode == BYTE)
		dump1(fp, mode);
	else
		dump2(fp, mode);
}
```

```
dump1(fp, mode)
FILE *fp;
int mode;
{
	int c = 0, offset, i;

	for (offset = 0; c != EOF; offset += 020) {
		printf("%07o", offset);
		for (i = 0; i < 16; i++) {
			if ((c = getc(fp)) == EOF)
				break;
			if (mode == ASCII) {
				if (iscntrl(c))
					switch(c) {
					case '\0':
						printf("  \\0");
						break;
					case '\b':
						printf("  \\b");
						break;
					case '\f':
						printf("  \\f");
						break;
					case '\n':
						printf("  \\n");
						break;
					case '\t':
						printf("  \\t");
						break;
					default:
						printf(" %03o", c);
						break;
					}
				else
					printf("   %c", c);
			} else   /* BYTE */
				printf("  %03o", c);
		}
	putchar('\n');
	}
}

dump2(fp, mode)
FILE *fp;
int mode;
{
	int word = 0, offset, i;

	for (offset = 0; word != EOF; offset += 020) {
		printf("%07o", offset);
		for (i = 0; i < 8; i++) {
			if ((word = getw(fp)) == EOF
				break;
			switch (mode) {
			case OCTAL:
				printf(" %06o", word);
```

```
                        break;
                case DECIMAL:
                        printf(" %05d", word);
                        break;
                case HEXDECI:
                        printf(" %04x", word);
                        break;
                default:
                        break;
                }
            }
            putchar('\n');
        }
}
```

grep

複数のファイルから文字列を検索

■Program Outline

grepはUNIXのコマンドのひとつで，複数のファイルの中から指定された文字列を含む行を検索する機能を持っています．

通常のこの種のユーティリティに比べ，複数のファイルを同時に検索できるので非常に有用です．

なお，ファイルを指定しない場合の入出力は，標準入出力となります．

■Explanation

EXEファイル名はgrep.EXEとします．

コマンドラインの入力の書式は以下のようになっています．

grep　オプション　文字列　ファイル名1〔ファイル名2………〕

オプションの内容については**表1**を参照してください．文字列は検索したい文字列を入れます．現在のバージョンではタブとスペースはサポートしていません．

ファイル名はOPTIMIZING Cの場合,（20－付加オプション数）個つづけることができます．

複数のファイルを指定した場合は，各ファイルの名前を表示してから，該当する行を出力します．

表1

オプション	機　　　能
－v	指定した文字列を含まない行を表示する
－n	表示する行の先頭にファイル中での行番号を表示する
－c	指定した文字列を含む行の行数のみを表示する
－l	指定した文字列を含むファイルのファイル名のみを表示する

■Attention

grepはこのあとに紹介するrplと大半の関数を共有しています．また内部処理上の都合で分割コンパイルした方が望ましいものがあるため，表2のようにソースファイルを三分割して，各々のコンパイル後リンクして下さい．

表2

ソース・ファイル名	含まれる関数名
grep.d	main, grep　(計2個)
match.d	Prepare, match, __match　(計3個)
sub.c	getexpre, check, escape, backet, closure, expand invent, squeeze, error　(計9個)

出力例(オプションを付けない場合)

```
A>grep char test1.c test2.c
test1.c:char *argv[];
test1.c:        char esc = 27;
test2.c:char *s;
test2.c:        char buf[256], *t = buf;
```

出力例(オプション－vを付けた場合)

```
A>grep -v char test1.c test2.c
test1.c:/* test program no.1 */
test1.c:main(argc, argv)
test1.c:int argc;
test1.c:{
test1.c:
test1.c:        while (--argc > 0)
test1.c:                printf("%c%s", esc, *++argv);
test1.c:}
test2.c:/* invert string */
test2.c:
test2.c:invert(s)
test2.c:{
test2.c:        int c = 1;
test2.c:
test2.c:        while (c < 0400)
test2.c:                *t++ = c++;
test2.c:        *t = '\0';
test2.c:        squeeze(buf, s);
test2.c:        strcpy(s, buf);
test2.c:}
```

■Source List

```
/*                                  */
/*     grep                         */
/*                                  */
/*     copyright (C) 1984           */
/*     programed by y.aihara        */
/*                                  */

#include <stdio.h>

#define MAXLEN 256
#define MAXEXPRES 32

typedef struct reg_exp {
     char *bf;
     int n, m;
} EXPRES;

static struct flags {
     unsigned invert : 1,
          number : 1,
          count  : 1,
          list   : 1,
          head   : 1;
} opts;

main(argc, argv)
int argc;
char *argv[];
{
     FILE *fp, *fopen();
     EXPRES expres[MAXEXPRES];
     int title, count;
     char *s;

     while (--argc > 0 && *argv[1] == '-')
          for (s = *++argv+1; *s; s++)
               switch (*s) {
               case 'v':
                    opts.invert = 1;
                    break;
               case 'n':
                    opts.number = 1;
                    break;
               case 'c':
                    opts.count = 1;
                    break;
               case 'l':
                    opts.list = 1;
                    break;
               case 'h':
                    opts.head = 1;
                    break;
               default:
                    argc = 0;
                    break;
```

```
            }
        if (argc < 1)
            error("Usage: grep [-vnclh] pattern file", NULL)
        escape(*++argv);
        if (prepare(*argv, expres) == -1)
            error("grep: bad pattern", NULL);
        if (argc == 1)  {
            count = grep(stdin, expres, NULL);
            if (opts.count)
                printf("%d\n", count);
        } else {
            title = argc > 2 && !opts.head;
            while (--argc > 0) {
                if ((fp = fopen(*++argv, "r")) == NULL)
                    error("grep: can't open %s", *argv);
                count = grep(fp, expres, title?*argv:NULL);
                if (opts.count && title ||\
                   opts.list && count) {
                    printf("%s", *argv);
                    putchar(opts.count ? ':' : '\n');
                }
                if (opts.count)
                    printf("%d\n", count);
                fclose(fp);
            }
        }
}

grep(fp, expres, name)
FILE *fp;
EXPRES *expres;
char *name;
{
     char line[MAXLEN], *match();
     int lineno, count;
     int len;

     lineno = count = 0;
     while (fgets(line, MAXLEN, fp)) {
          lineno++;
          line[strlen(line)-1] = '\0';
          if (!match(line, expres, &len) == opts.invert) {
               count++;
               if (opts.count || opts.list)
                    continue;
               if (name)
                    printf("%s:", name);
               if (opts.number)
                    printf("%d:", lineno);
               puts(line);
          }
     }
     return(count);
}
```

```
/* match */

#define MAXLEN 256
#define MAXEXPRES 32
#define NULL 0
#define DOT 0

typedef struct reg_exp {
	char *bf;
	int n, m;
} EXPRES;

static struct flags {
	unsigned head : 1,
		tail : 1;
} flg;

prepare(pttn, expres)
char *pttn;
EXPRES expres[];
{
	char *p, *check(), *getexpres();
	int i;

	if (*pttn == '^') {
		flg.head = 1;
		pttn++;
	}
	if (!(p = check(pttn)))
		return(-1);
	if (*p == '$') {
		flg.tail = 1;
		*p = '\0';
	}
	for (i = 0; i < MAXEXPRES; i++)
		if (!(pttn = getexpres(pttn, &expres[i]))) {
			expres[i].bf = NULL;
			return(0);
		}
	return(-1);
}

char *match(srch, expres, len)
char *srch;
EXPRES *expres;
int *len;
{
	char *p, *_match();

	do {
		if ((p = _match(srch, expres)) &&\
			(!flg.tail || !*p)) {
			*len = p - srch;
			return(srch);
		}
	} while (*srch++ && !flg.head);
```

```
        return(NULL);
}

char *_match(srch, expres)
char *srch;
EXPRES *expres;
{
        char *p, *_match();
        int i;
        if (!expres->bf)
                return(srch);
        for (i = 0; i < expres->n; i++) {
                if (!*srch ||\
                *expres->bf != DOT && !index(expres->bf, *srch))
                        return(NULL);
                srch++;
        }
        for ( ; i <= expres->m; i++) {
                if ((p = _match(srch, expres + 1)) &&\
                    (!flg.tail || !*p))
                        return(p);
                if (!*srch ||\
                *expres->bf != DOT && !index(expres->bf, *srch))
                        return(NULL);
                srch++;
        }
        return(NULL);
}




/* get expression */

#define NULL 0
#define DOT 0

typedef struct reg_exp {
        char *bf;
        int n, m;
} EXPRES;

char *getexpres(pttn, expres)
char *pttn;
EXPRES *expres;
{
        char buf[256], *p = buf;
        char *malloc(), *bracket(), *closure();

        if (!*pttn)
                return(NULL);
        else {
                switch (*pttn) {
                case '\\':
```

```
                pttn++;
                *p++ = *pttn++;
                *p = '\0';
                break;
        case '.':
                pttn++;
                *p++ = DOT;
                break;
        case '[':
                pttn = bracket(pttn, p);
                break;
        default:
                *p++ = *pttn++;
                *p = '\0';
                break;
        }
        expres->bf = malloc(strlen(buf)+1);
        strcpy(expres->bf, buf);
        switch (*pttn) {
        case '{':
        case '*':
        case '+':
        case '?':
                pttn = closure(pttn, expres);
                break;
        default:
                expres->n = expres->m = 1;
                break;
        }
        return(pttn);
    }


/* check */

#define NULL 0

char *check(pttn)
char *pttn;
{
        int t = 0;

        while (*pttn)
                switch(*pttn++) {
                case '\\':
                        if (!*pttn++)
                                goto err;
                        t = 1;
                        break;
                case '[':
                        if (*pttn == '^')
                                pttn++;
                        if (*pttn == ']')
                                pttn++;
                        while (*pttn && *pttn != ']')
```

```
                    pttn++;
                if (!*pttn++)
                    goto err;
                t = 1;
                break;
            case '{':
                if (!t)
                    goto err;
                while (isdigit(*pttn))
                    pttn++;
                if (*pttn == ',') {
                    pttn++;
                    while (isdigit(*pttn))
                        pttn++;
                }
                if (*pttn++ != '}')
                    goto err;
                t = 0;
                break;
            case '*':
            case '+':
            case '?':
                if (!t)
                    goto err;
                t = 0;
                break;
            case ']':
            case '}':
                goto err;
            default:
                t = 1;
                break;
            }
    return(--pttn);
err:
    return(NULL);
}


/* escape sequence */

escape(s)
char *s;
{
    char *t = s;
    char bf[4], *p;
    int i;

    while (*s)
        if (*s == '\\')
            switch (*++s) {
            case 'n':
                *t++ = '\n';
                s++;
                break;
            case 't':
```

```
                    *t++ = '\t';
                    s++;
                    break;
                case 'r':
                    *t++ = '\r';
                    s++;
                    break;
                case 'f':
                    *t++ = '\f';
                    s++;
                case 's':
                    *t++ = ' ';
                    s++;
                    break;
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                    p = bf;
                    for (i = 0; i < 3; i++) {
                        if (*s < '0' || *s > '7')
                            break;
                        *p++ = *s++;
                    }
                    *p = '\0';
                    *t++ = otoi(bf);
                    break;
                default:
                    *t++ = '\\';
                    *t++ = *s++;
                }
            else
                *t++ = *s++;
    *t = '\0';
}

otoi(s)
char *s;
{
    int n = 0;

    while (*s >= '0' && *s <= '7')
        n = 8 * n + *s++ - '0';
    return(n);
}


/* bracket */

char *bracket(pttn, str)
char *pttn, *str;
{
    char *p = str;
```

```
        int inv = 0;

        if (*++pttn == '^') {
                inv = 1;
                pttn++;
        }
        if (*pttn == ']')
                *p++ = *pttn++;
        while (*pttn != ']')
                *p++ = *pttn++;
        *p = '\0';
        expand(str);
        if (inv)
                invert(str);
        return(++pttn);
}

/* closure */

#define NULL 0

typedef struct reg_exp {
        char *bf;
        int n, m;
} EXPRES;

char *closure(pttn, expres)
char *pttn;
EXPRES *expres;
{
        switch (*pttn) {
        case '{':
                expres->n = atoi(++pttn);
                expres->m = 0;
                while (*pttn != '}')
                        if (*pttn++ = ',' &&\
                            (expres->m = atoi(pttn)) <= 0)
                             expres->m = 0400;
                if (!expres->m)
                        expres->m = expres->n;
                break;
        case '*':
                expres->n = 0;
                expres->m = 0400;
                break;
        case '+':
                expres->n = 1;
                expres->m = 0400;
                break;
        case '?':
                expres->n = 0;
                expres->m = 1;
                break;
        }
        return(++pttn);
}
```

```
/* expand */

expand(s)
char *s;
{
     char c, buf[256];
     char *p = s, *q = buf;

     while (*p) {
          *q++ = *p++;
          if (*p == '-' && *(p-1) <= *(p+1)) {
               c = *(p-1)+1;
               while (c <= *(p+1))
                    *q++ = c++;
               p += 2;
          }
     }
     *q = '\0';
     strcpy(s, buf);
}

/* invert */

invert(s)
char *s;
{
     char buf[256], *t = buf;
     int c = 1;

     while (c < 0400)
          *t++ = c++;
     *t = '\0';
     squeeze(buf, s);
     strcpy(s, buf);
}

/* squeeze */

squeeze(s, t)
char *s, *t;
{
     char *p, *q;

     for (p = s; *p; p++) {
          for (q = t; *q; q++)
               if (*p == *q)
                    break;
          if (!*q)
               *s++ = *p;
     }
     *s = '\0';
}
```

```
/* error masage print and exit */

#include <stdio.h>

error(s1, s2)
char *s1, *s2;
{
        fprintf(stderr, s1, s2);
        fprintf(stderr,"\n");
        exit(1);
}
```

rpl

文字列の置き換えとフィルタの機能

■Program Outline

このコマンドは，ファイル内の文字列を交換するものです．MS-DOS のエディタにも replace コマンドがありますが，rpl コマンドを使えばもっと簡単に交換することができます．また，フィルタとしても使えます．この rpl は grep と同じパターン・マッチング能力を持っています．

■Explanation

このコマンドの書式は次の通りです．

```
rpl  from  to 〔<in〕 〔>out〕
```

MS-DOS では，″や’をデミリタとして使用できないので，空白やタブをパターンの中に含めることはできません．しかし，これではこのコマンドの能力は半減してしまうのでエスケープ・シーケンスを使えるようにしました．

これによって復改，タブ，空白などは￥記号に続く n，t，s で代用できます．また，これ以外のコントロールコードなど非印字文字も￥記号に続く1～3桁の8進数で表わすことができます．

このコマンドを使うとき注意することは，そのパターン・マッチング能力が強力なため，思わぬ副作用が起こる危険性があることです．安全策はファイルにリダイレクト出力する前に，コンソールに出力させて確認することです．

ファイルを圧縮する方法で，簡単なためよく使われるものに，連続した空白をタブで置き換える方法があります．これはたとえば，

```
rpl  ￥s{8}  ￥t  <old_file  > new_file
```

として使えますが，この圧縮ファイルがもと通りに展開される保証はないので，実行しない方がよいでしょう．むしろ，タブが8文字の空白に展開されるとC

のプログラムソースが読みにくいので，タブを5文字くらいの空白に置き換える方が有効でしょう．こういう短いBATCHファイルを自分流に作っておくとよいと思います．

rpl　¥t　¥s¥s¥s¥s¥s　<%1

他に有用な例としては，テキスト行の余分な空白およびタブを取り除く例や，

rpl　〔¥s¥t〕+$　¥0　<%1

プリンタに出力するとき左マージンを指定する例があります．

rpl　^　¥t　<%1

次にパイプを使った例では，英文のテキストファイルの単語の出現頻度を表示させる例をあげます．

rpl　〔¥s¥t.,;:〕+¥n<%1 ¦ grep－v ^$ ¦ sort ¦ uniq－c

まず，空白，タブ，ピリオド，コロンなどを復改に置き換えます．そして，空白行を削除してsortを行い最後に重複回数を数えて表示しています．

この他，いろいろな組み合わせでちょっとしたユーティリティが作れると思いますので，作ってみてはいかがでしょう．

また，第2パラメータには正規式は使えません．

■Attention

rplはこのgrepとほとんど同一の関数から成り立っていますので，新しくソースファイルを作るのはrplのmain関数だけですみます．

grepのソースファイルのうち，match.cとsub.cはそのまま使い，各々コンパイル後grepのmainとリンクしてください．

ソース・ファイル名	含まれる関数名
rpl.c	main(rplのもの)――新規作成
match.c	grepのファイルと同内容
sub.c	〃

■Source List

```
/*                                    */
/*    replace                         */
/*                                    */
/*    copyright (C) 1984              */
/*    programed by y.aihara           */
/*                                    */

#include <stdio.h>

#define MAXLEN 256
#define MAXEXPRES 32

typedef struct reg_exp {
     char *bf;
     int n, m;
} EXPRES;

main(argc, argv)
int argc;
char *argv[];
{
     EXPRES expres[MAXEXPRES];
     char line[MAXLEN], *s, *t, *match();
     int head = 0;
     int len;

     if (argc != 3)
          error("Usage: rpl from to", NULL);
     if (*argv[1] == '^')
          head = 1;
     escape(argv[1]);
     escape(argv[2]);
     if (prepare(argv[1], expres) == -1)
          error("rpl: bad pattern", NULL);
     while (gets(line, MAXLEN, stdin)) {
          for (s=line; t=match(s,expres,&len); s+=len)
               while (s < t)
                    putchar(*s++);
               printf("%s", argv[2]);
               if (!*s || head)
                    break;
          }
          puts(s);
     }
}
```

wc

ファイルの行数，単語数，文字数を表示

■Program Outline

wcとはWord Countの頭文字を取ったものです．このwcは，指定したファイルの行数，単語数，文字数を数えるためのプログラムです．使い方はUNIXとほぼ同じですが，ワイルドカードは使えません．

■Explanation

ファイルネームはwcとして実行して下さい．コマンドラインからの入力は次のようになります．

wc 〔－lwc〕〔path-name〕………

－lwcはオプションで，機能は右表のようになっています．このオプションを省略すると，－lwcの全部を指定したことになります．

オプション	機　能
l	行数の指定
w	単語数の指定
c	文字数の指定

path-nameはどのファイルを調べるのかを指定します． このpath-nameを続けていくつも入力した場合は，入力したファイルのすべてを調べます．また，これを省略すると標準入力パスになります．

出力例

```
OS9:wc where.c sort.c wc.c
 75    169   1753 where.c
180    411   3961 sort.c
 92    198   1855 wc.c
```

■Source List

```
/*                                  */
/*          Word Count              */
/*                                  */
/*       Copyright (C) 1984         */
/*     Programed by M.ICHIDA        */
/*                                  */

#include <stdio.h>      ────標準入出力ヘッダをインクルードする
#define TRUE   1
#define FALSE  0
#define ON     1
#define OFF    0

main(argc,argv)
int argc;
char *argv[];
{
  int line,word,chr;
  line=word=chr=ON;

  ++argv;
  if (--argc > 0)
    if (argv[0][0] == '-')──第1パラメタの先頭が“―”ならばすべてのフラグを
      {                                                        OFF にする
      line=word=chr=OFF;
      --argc;
      while (strlen(++argv[0]) > 0)  ┐
        switch (argv[0][0])          │
          {                          │
          case 'l':                  │
            line = ON;               │
            break;                   │ パラメタが L，W，C ならばそれに応じ
          case 'w':                  │              たフラグを ON とする
            word = ON;               │
            break;                   │
          case 'c':                  │
            chr = ON;                │
            break;                   │
          default:                   ┘
            printf("\n Use: Wc [-l,w,c] [<path>]\n");  ┐
            printf("\n  Count lines,words and");       │ パラメタが不
            printf(" charactors in file\n");           │ 完全な場合は
            printf("         l:count lines\n");        │ その方法を示
            printf("         w:count words\n");        │ 唆する
            printf("         c:count charactors\n");   ┘
            exit(0);
            break;
          }
      ++argv;
      }
  if (argc == 0)
    count(stdin,line,word,chr);  ────カウントルーチン
  else
    while (argc-- > 0)
```

```
        {
          count(fopen(*argv,"r"),line,word,chr);——各ファイル名における
          printf(" %s\n",*argv++); ———————ファイル名を    カウントルーチン
        }                                       表示
}
count(file,lf,wf,cf)
FILE *file;
int lf,wf,cf;
{
  int c,line,word,chr,flag = FALSE;
  line=word=chr=0;
  if (file == NULL)
    exit(errno);
  while ((c = getc(file)) !=EOF)
    {
      ++chr; ————————文字数のカウント
      if (c == '\n')
        ++line; ————————リターンコードのカウント
      if (c == ' '
        ||c == '\n'
        ||c == '\t'
        ||c == '\l' )
        flag = FALSE;
      else if (flag == FALSE)
        {
          ++word; ————————ワードのカウント
          flag = TRUE;
        }
    }
  if (lf == ON)
    printf("%7d",line);  ┐
  if (wf == ON)          │
    printf("%7d",word);  │ 各カウンタ値の表示
  if (cf == ON)          │
    printf("%7d",chr);   ┘
}
```

where

ファイルの中のディレクトリを探し出す

■Program Outline

このプログラムは，UNIXのfindコマンドを簡略化したもので，必要なファイルのフルパスネームを求めてくれます．つまり，あるファイルをアクセスしたいときなどには，そのファイルがどのディレクトリにあるのか探すことができます．

■Explanation

まずwhereというファイルネームで実行可能な状態(ロードするか，あるいはカレント実行ディレクトリに入れる)にしてください．

コマンドラインからの入力は次のようになります．

where 〔directry-path〕 file-name

〔directory-path〕とはどのディレクトリを探すのか，ということで，ここでディレクトリを指定した場合に，そのディレクトリの下にあるすべてのファイルを調べます．この〔directory-path〕を省略すると，カレントディレクトリとなります．

file-nameは目的の探したいファイルネームです．

このファイルネーム中に？という文字を入れた場合には，その文字はどんな文字でもよい，ということを意味します．

たとえばUNI？とUNIXでもUNI+でもUNI－でもよいということになります．このような文字をワイルドカードと呼んでいます．

出力例

```
OS9:where /d1 m???????
/d1/newBoot/MD0
/d1/newBoot/MD1
/d1/CMDS/Makdir
/d1/CMDS/Mdir
/d1/CMDS/Merge
/d1/CMDS/Mfree
/d1/SYS/Motd
/d1/DEFS/SOURCE/MD0
```

■Source List

```
/*                                  */
/*            Where                 */
/*                                  */
/*       Copyright (C) 1984         */
/*     Programed by  M.ICHIDA       */
/*                                  */

#include <stdio.h> ―――――標準入出力ヘッダをインクルードする
#include <direct.h> ――――ディレクトリ処理に関するヘッダをインクルードする
#include <ctype.h> ―――――文字処理に関するヘッダをインクルード
#define MAX_DIR_NAME 80
#define MAX_NAME      30
#define DIRECTORY  0x81

main(argc,argv)
int argc;
char *argv[];
{
  char dirc[MAX_DIR_NAME],name[MAX_NAME];
  struct dirent buffer;
  if ((argc > 3 ) || (argc < 2))
     {
       printf("\n Use: Where [<Directory-path>]");     ┐コマンドラインが
       printf(" <file-name>\n");                        │不適当ならその入
       printf("\n This command is to search");          │力方法を示唆する
       printf("  <file-name> under <Dir-path>.\n");    ┘
       exit(0);
     }
  strcpy(dirc,(argc == 3) ? *++argv : ".");―――ディレクトリをセット
  strcpy(name,*++argv); ――――探したいファイル名をセット
  search(dirc,name); ―――――ディレクトリ中のファイル名の検索
}

search(name,file)
char name[],file[];
{
  int i,opn;
  char *po,*pos;
  struct dirent buffer;
  if ((opn=open(name,DIRECTORY)) == -1) exit(errno);――――ディレ
  pos=name+strlen(name);――ポインタposをnameの最後に        クトリファイル
  *pos++='/'; ――――――nameの最後に"/"をつけ足す              のオープン
  while(read(opn,&buffer,32) > 0) ――――バッファにデータ(パスネーム)
    {                                                        を読み込む
      *pos='\0'; ――――――nameの最後にヌルコードをつけ足す
      if (buffer.dir_name[0] == NULL) ――――デリートされたファ
              continue;                                  イルなら次へ
      for (po=buffer.dir_name;(*po & 0x80) == 0;po++); MSBが1の
      *po&=0x7f; ――――――MSBをクリアする                        所まで探す
      *++po='\0'; ――――――ファイル名の最後にヌルコードをつけ足す
      if (     strcmp(buffer.dir_name,".")  == 0
           || strcmp(buffer.dir_name,"..") == 0 )――もしファイルが
                 continue;                              "."","".." なら次へ
      strcat(name,buffer.dir_name); ――――パスリストの追加
```

```
        if ( strcomp(buffer.dir_name,file) == 0 )―指定したファイル名か？
               printf("%s\n",name); ――――― YESなら表示
        i=access(name,DIRECTORY); ――――ディレクトリファイルが開けるか？
        if (i==0)
             search(name,file); ――――次の探したいディレクトリを探す
    }
  *--pos='\0'; ――――――ディレクトリ名をもとに戻す
  close(opn);  ――――――ファイルのクローズ
}

strcomp(c1,c2)
char *c1,*c2;
{
  do
    if (*c2 != '?')
      if (toupper(*c1) != toupper(*c2)) return(-1);――文字の比較：異
  while((*c1++ != '\0') && (*c2++ != '\0'));―どちらか　なればー1を返す
  return(0);                                  が終るまでくり返し
}
```

エラーメッセージ

MICROWARE OS-9 C v.1.1.4にはいくつかのバグや，使いにくい点があります．たとえば，浮動小数点数(float)やlongをprintfで使うときにpffinit( )やpflinit( )を入れなければならないことなどは，マニュアルに書いてあるのでよいとしても，関数の中で配列を大きく取ったときなど(mainも含む)，実行時に**stack overflow**となってしまいます．たまには暴走してしまうこともあるようです．

これは，実行時に＿＿＿＿#5kなどのようにメモリを大きく割り当てれば使えます．またコンパイル時に“－m”オプションを使って大きなメモリを指定すればうまく動いてくれます．これなどは，素人にはなんで動いてくれないのかわからないことがあると思いますので，ぜひ改良してほしいと思います．このほかには，chmodという関数が動いてくれないので困っています．あとgetchar( )をgetc(stdin)に置き換えるはず(stdio.hの中にそう定義されている)なのになぜかgetc(stdin)( )とカッコがよけいについてしまいます．これは自分で置き換えてしまえばよいのですが………．

head

ファイルを行指定で出力する

■Program Outline

このプログラムは，ファイルの先頭部分を出力するためのものです．単にファイルの最初から出力するだけではありません．ファイルの先頭から何行目を始めとして何行分出力するのかも指定することができます．

■Explanation

ファイルネームをheadとして実行してください．コマンドラインからの入力は次のようになります．

head 〔＋n〕〔－m〕〔path-name〕

＋nで何行目から出力するのかを指定します．

－mで何行分出力するかを指定します．ここで，－＊とすると最後まで出力するという意味になります．

path-nameで入力ファイルを指定します．省略すれば標準パスになります．

例　head　＋5　－20　test.1　（この場合，test.1というファイルの先頭の5行目から20行分を出力します）

出力例

```
OS9:head -15 head.c
/*                                  */
/*              Head                */
/*                                  */
/*      Copyright (C) 1984          */
/*    Programed by M.ICHIDA         */
/*                                  */

#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
  char buffer[BUFSIZ];
  int i,line = 10,head = 1,toeof = 0;
```

■Source List

```
/*                            */
/*          Head              */
/*                            */
/*    Copyright (C) 1984      */
/*   Programed by M.ICHIDA    */
/*                            */

#include <stdio.h> ――――――――標準入出力ヘッダのインクルード

main(argc,argv)
int argc;
char *argv[];
{
  char buffer[BUFSIZ];
  int i,line = 10,head = 1,toeof = 0; ――――――――デフォルト値の設定

  while (--argc > 0)
      switch ((*++argv)[0])
        {
        case '-':
          if (argv[0][1] == '*') ――――――――パラメタが“―＊”ならば toeof
            {                                  フラグを ON とする
              toeof = 1;
              break;
            }
          else
            {
              line = atoi(*argv + 1);――――――――パラメタが“―”ならばそ
              break;                           の次の数を Line に代入する
            }
        case '+':
          head = atoi(*argv + 1); ――――パラメタが“＋”ならばその次の数を
          break;                                   head に代入する
        default:
          if (freopen(argv[0],"r",stdin) == NULL)―パラメタがファイ
            exit(errno);                         ル名ならばそのフ
        }                                        ァイルをリードモ
  for (i = 1;i < head;i++)                       ードでオープン
    if (gets(buffer) == NULL) ――――――――ファイルポインタを head 個進める
      exit(0);
  while ( line-- > 0 ||
          toeof ==1      )
    {
      if (gets(buffer) == NULL)
        exit(0);
      printf("%s\n",buffer); ――――――――バッファの内容を表示
    }
  while (gets(buffer) != NULL)――――ヌルが出現するまでファイルから読み込む
    ;
}
```

tail

ファイルの最後の部分を出力

■Program Outline

前述のheadはファイルの最初の部分を出力するプログラムですが，反対にこのtailはファイルの最後の部分を出力するためのプログラムです．このtailは，ファイルの後ろから何行分出力させるかを指定して使います．

■Explanation

このプログラムは，ファイルネームをtailとして実行します．
コマンドラインからの入力は次のようになります．

tail 〔−n〕〔path-name〕

−nによりファイルの終わりから何行分出力するかを指定します．省略すると，−nは−10に設定されます．

path-nameは入力ファイルの指定です．これを省略した場合な標準入力パスになります．

プログラムを作っているときなど，プログラムの最後に付けたルーチンがおかしいということがたまにあります．そのプログラムが数百行にもわたる場合には，最後のほんの数行を調べるために全リストを見るのはめんどうなものです．そんなときに，このtailは役に立ちます．

例　　tail　−5　tail.c（この場合，tail.cというファイルの最後の20行を出力）

出力例
```
OS9:tail -5 tail.c
    }
  for (i = 0;i <= tail;i++)
    printf("%s\n",string[i]);
  fclose(infile);
}
```

■Source List

```
/*                              */
/*          Tail                */
/*                              */
/*      Copyright (C) 1984      */
/*    Programed by M.ICHIDA     */
/*                              */

#include <stdio.h>
#define MEMERR 207

main(argc,argv)
int argc;
char *argv[];
{
  char **string;
  char buffer[BUFSIZ];
  int i,len,tail=10;
  FILE *infile = stdin;

  ++argv;
  if (--argc != 0)
    if (argv[0][0] == '-')
      {
        tail = atoi(*argv++ + 1);
        --argc;
      }
  if (argc != 0)
    if ((infile = fopen(argv[0],"r")) == NULL)
      exit(errno);
  if ((string = (char **) calloc(tail,sizeof(char *))) == NULL)
    exit(MEMERR);
  --tail;
  for (i = 0;i <= tail;i++)
    {
      if ((string[i] = malloc(1)) == NULL)
        exit(MEMERR);
      string[i][0] = '\0';
    }
  while (fgets(buffer,BUFSIZ,infile) != NULL)
    {
      free(string[0]);
      for (i = 0;i < tail;i++)
        string[i] = string [i+1];
      len = strlen(buffer);
      buffer[len-1] = '\0';
      if ((string[tail] = malloc(len)) == NULL)
        exit(MEMERR);
      strcpy(string[tail],buffer);
    }
  for (i = 0;i <= tail;i++)
    printf("%s\n",string[i]);
  fclose(infile);
}
```

sort

行単位やフィールド別に自在なソーティング

■Program Outline

これはファイルの内容を行単位でソート（並び換える）するコマンドです．UNIXの同名のコマンドに可能なかぎり近付けたものですが，オプションなどでいくつかサポートされていないものがあります．

■Explanation

このプログラムは，sortというファイルネームで実行します．コマンドラインからの入力は次のようになります．

sort 〔－frb〕〔path-name〕〔＋n〕〔－o output-path〕

－frbはオプションです．これらのオプションは－frや－rbなどのように組み合わせて使うことができます．

オプション	機　　能
f	大文字と小文字の区別をしない
r	並び換える順を逆にする
b	ブランク文字(スペース,タブ等)を無視する

path-nameはsortするファイルのパスネームで，省略すると標準入力パスからの入力をsortします．

＋nはsortするときにどのフィールドからを比較の対象にするのかを指定します．フィールドとは1つ以上のスペースまたはタブで区切られたものです．

例　abcdef␣ghij␣k␣␣ℓmn

第0フィード　第1フィールド　第2フィールド　第3フィールド

ここで，たとえばオプション＋nを＋2とすると，上の例において第2フィールドつまりk以降を比較の対象とします．オプション＋nを指定しない場合は，＋nは0にセットされ，ファイルの最初（第0フィールド）からが比較の対象になります．

－o　output-pathは，ソートしたファイルを出力したいときに，そのファイル名を指定します．そのファイル名を省略すると標準出力パスに出力します．

例　　－o　test.1

また，sort　－o　test.1 などは，sort＞test.1のようにリダイレクトしたものと同じです．

次に，sortと他のコマンドをパイプラインした例を示します．

例　　dir　e　！　head　＋3　－＊　！　sort　＋6　－f（ABC順で並び換え）
　　　dir　e　！　head　－3　－＊　！　sort　＋1（ファイルを古い順に並び換え）

ソートするサンプルテキスト

```
OS9:list test
SEIKO MATSUDA
AKINA NAKAMORI
KYOKO KOIZUMI
NAOKO KAWAI
HIDEMI ISHIKAWA
HIROKO YAKUSHIMARU
IYO MATSUMOTO
YOSHIE KASHIWABARA
TOMOYO HARADA
YOSHIMI IWASAKI
```

出力例（ABC順）

```
OS9:sort test
AKINA NAKAMORI
HIDEMI ISHIKAWA
HIROKO YAKUSHIMARU
IYO MATSUMOTO
KYOKO KOIZUMI
NAOKO KAWAI
SEIKO MATSUDA
TOMOYO HARADA
YOSHIE KASHIWABARA
YOSHIMI IWASAKI
```

出力例（逆ABC順）

```
OS9:sort -r test
YOSHIMI IWASAKI
YOSHIE KASHIWABARA
TOMOYO HARADA
SEIKO MATSUDA
NAOKO KAWAI
KYOKO KOIZUMI
IYO MATSUMOTO
HIROKO YAKUSHIMARU
HIDEMI ISHIKAWA
AKINA NAKAMORI
```

■Source List

```
/*                                 */
/*            Sort                 */
/*                                 */
/*      Copyright (C) 1984         */
/*    Programed by M.ICHIDA        */
/*                                 */

#include <stdio.h>
#include <ctype.h>
#define MAXLINES 1000
#define TOOLONG 207
#define ON  1
#define OFF 0

char *lineptr[MAXLINES];

main(argc,argv)
int argc;
char *argv[];
{
  int rev=1,strp=0,lines,blankskip = OFF,
      strcmp(),strcomp(),(*scmp)() = strcmp;
  FILE *outfile = stdout,*infile = stdin;
  extern char *lineptr[];

  while (--argc > 0)
    {
      switch ((*++argv)[0])
      {
        case '+':
          strp = atoi(*argv + 1);
          break;
        case '-':
          while(argv[0][1] != '\0' &&
                argv[0][1] != ' '       )
            switch ((++argv[0])[0])
              {
              case 'r':
                rev = -1;
                break;
              case 'o':
                outfile = fopen(*++argv,"w");
                break;
              case 'b':
                blankskip = ON;
                break;
              case 'f':
                scmp = strcomp;
                break;
              default:
       printf("\n Use: Sort [<path>] [-f,r,b]");
               printf(" [+n] [-o <output-path>]\n");
               printf("\n  This command is to sort <path>");
               printf(" or <std-input>.\n");
```

```
                printf("    -f:prohibit to distinguish");
                printf(" upper");
                printf(" and lower case\n");
                printf("    -r:reverse sorting sequence\n");
                printf("    -b:skip blanks and tabs\n");
                printf("    +n:set sorting start field");
                printf(" (n:digit)\n");
                printf("    -o:specify output path\n");
                exit(0);
                break;
                }
            break;
          default:
            if ((infile = fopen(argv[0],"r")) == NULL)
              exit(errno);
            break;
        }
    }
  if ((lines = readlines(infile)) >= 0 )
    {
      sort(lines,rev,strp,blankskip,scmp);
      writelines(lineptr,lines,outfile);
    }
  else
     exit(TOOLONG);
}


readlines(infile)
FILE *infile;
{
  extern char *lineptr[];
  int nlines = 0;
  char *p,line[BUFSIZ];
  unsigned len;

  while (fgets(line,BUFSIZ,infile) != NULL)
   {
    len=strlen(line);
    if (nlines >= MAXLINES)
      return(-1);
    else if ((p = malloc(len)) == NULL)
      return(-1);
    else
      {
        line[len-1] = '\0';
        strcpy(p,line);
        lineptr[nlines++] = p;
      }
   }
  return(nlines);
}

sort(n,rev,strp,skip,scmp)
int n,rev,strp,skip,(*scmp)();
{
```

```
    extern char *lineptr[];
    int gap,i,j;
    char *temp,*pointa,*pointb,*fieldset(),*blankskip();

    for (gap = n/2;gap > 0;gap /=2)
      for (i = gap;i < n;i++)
        for (j = i- gap;j >= 0;j -= gap)
          {
            pointa = fieldset(lineptr[j],strp);
            pointb = fieldset(lineptr[j+gap],strp);
            if (skip == ON)
              {
                pointa = blankskip(pointa);
                pointb = blankskip(pointb);
              }
            if (((*scmp)(pointa,pointb) * rev) <= 0)
              break;
            temp = lineptr[j];
            lineptr[j] = lineptr[j+gap];
            lineptr[j+gap] = temp;
          }
}

writelines(ptr,nlines,outfile)
char *ptr[];
int nlines;
FILE *outfile;
{
  while (--nlines >= 0)
    fprintf(outfile,"%s\n",*ptr++);
}

char *fieldset(point,fld)
int fld;
char *point;
{
  int i;

  for (i = 0;i < fld;i++)
    {
      if (*point == ' ')
        while(*point++ == ' ')
          ;
      while(*point != ' '  &&
            *point != '\t' &&
            *point != '\0'   )
        point++;
    }
  return(point);
}

char *blankskip(point)
char *point;
{
  while(*point == ' ' ||
        *point == '\t'  )
```

```
    ++point;
  return(point);
}

strcomp(s1,s2)
char *s1,*s2;
{
  int i;
  char c1,c2;

  do
    {
      c1=toupper(*s1);c2=toupper(*s2);
      if (c1 != c2)
        return((c1 < c2) ? -1 : 1);
    }
  while ((*s1++ != '\0') && (*s2++ != '\0'));
  return(0);
}
```

screen

キャラクタのハードコピーコマンド(F-BASIC Ver.3.0に相当)

■Program Outline

このscreenは，画面に表示されているキャラクタを，プリンタ等にハードコピーするためのプログラムです．

F-BASICのVersion3.0までのHARDC 0に相当するものです．ハードコピーの取れる範囲は，キャラクタで80×25の大きさ(フルサイズ)のウィンドゥに限られ，Version4.0以上のコピー範囲の指定の機能などはありません．その代わりに，OS-9のリダイレクト機能を生かして，プリンタだけでなくディスクファイルにも出力させることができます．

■Explanation

ファイルネームはscreenです．

screenにはオプションはありません．その代わり，ディスクファイル等に画面内容を出力できるなど，画面内容の出力デバイスを内部で固定せずに，標準出力パスに出力するようになっています．普通，標準出力パスは/Term(CRT)に対して開かれていますから，このままではコピーした画面内容はまた画面に対して出力されて，いたずらに画面を乱すだけです．そこで実際に使用するときには，標準出力を画面内容を出力させたいデバイス，ディスクファイルにリダイレクトさせて使うことになります．

コマンドラインの書式は次のようになります．

screen　＞**パスリスト**

パスリストにはI/Oデバイスのパスリスト，ディスクファイルのパスリストが入ります．たとえば，プリンタに画面データを出力させようとした場合（通常のハードコピー)には，FM-7のOS-9プリンタのパスリストは/Pですから，

screen　＞/P

とコマンドを投入します．

現在のデータディレクトリ上のcrt＿copyというディスクファイルに出力させたい場合には，

```
screen  >crt_copy
```

となります．

以下に**出力例**を示します．この**出力例**は，ソースプログラムscreen.cをcc1でコンパイルしたあとで，作成されたオブジェクトに対して，Identコマンドを使用した画面をプリンタに出力させたものです．

■Attention

プログラムは簡単なものです．mainのみで成り立っています．FM-7(FMシリーズ)のOS-9に用意されたデータの入出力でサブシステムをコントロールするという方法で，サブシステムコマンドGet Character Block1により画面1行分のデータを得て、それを必要な部分だけ出力しています．

出力例

```
OS9:cc1 screen.c
cc1 version 1.1.4
Copyright 1983 Microware
'screen.c'
c.prep:
c.pass1:
c.pass2:
c.opt:
c.asm:
c.link:

OS9:ident screen -xs
   1 $11 $523F43 . screen

OS9:screen >/P
```

サブシステムをコントロールするためのデータを送るパスとして，パス番号0＝標準入力パスを使用しています．一般に，サブシステムをコントロールするためのデータは，標準出力パスに送られる(BASIC 09用のグラフィックコマンドなどはこのようになっています)のですが，標準出力パスを内容の出力先のリダイレクト用にするためにこうしました．したがって，標準入力をリダイレクトすることはできません．

データを送り出すための構造体変数，受け取るための構造体変数が外部変数として宣言されていますが，これは関数mainの中身を少しでも短くしようとしただけのことで他意はありません．また，変数のすべてにdirectという宣言がなされていますが，これはマイクロウェア社のCにだけある特別のクラスであり，変数がダイレクトページ内に置かれることによりアクセスが速くなるのです．すべての変数を自動変数にしても，まったく同様の動作をします．

Source List

```
/*                                          */
/*                Screen                    */
/*                                          */
/*        Copyright (C) 1984                */
/*      Programed by H.Hatayama             */

#include <stdio.h>

#define DATSIZ 80


direct struct subpack
{
    /* os9 sub-packet head */
    char head, sw, insz ,outsz;

    /* FM subsystem command */
    char dam1, dam2, cmdcd, x1, y1, x2, y2;

    /* os9 sub-packet end */
    char etx;

} os9rcb = {
                0x14,0x0f,DATSIZ+4,7,
                0,0,6,0,0,79,0,
                3
            };  /* for send data to sub */

direct struct subin
{
    char head2,sw2,insz2,outsz2;

    char errcd,cntf,dammy,dtsz,buff[DATSIZ];

    char etx2;

} subdata;  /* for get data from sub */



/*                                      */
/*      main routine of screen          */
/*                                      */

main()
{
    static direct int x,y,data_no;
    static direct int stdin_=0,stdout_=1;
    static direct char line[80];

    for ( y=0; y<=24; y++) {

        /* Set subrcb */
        os9rcb.y1=os9rcb.y2=y;
```

```
        /* Get character data of 1 line pointed by y */
        data_no = write(stdin_,&os9rcb,sizeof(struct subpack));
        data_no =  read(stdin_,&subdata,sizeof(struct subin));

        /* make screen image line until find NULL(00) in buffer */
        for(x=0;
            (line[x] = subdata.buff[x]) != NULL && x<=79;x++);

        printf("%s\n",line);
    }
}
```

エラーメッセージ

MICROWARE OS-9 C v.1.1.4のヘッダファイルはなかなかよくできているので便利なのですが，中には使い方のわからないものなどがあります．たとえば <direct.h> などは，最初の構造体direntはディレクトリファイルのデータの形を表しているのはわかりますが，その他のものはCの上からどのようにアクセスしたら手に入れることができるか，データの形がどのように定義されているのかわからないものがあります．詳しいことを書いてほしかったと思います．

ところで，ヘッダファイルにもバグがありました．それは <sgstat.h> というヘッダで，getstat，setstatという関数で使われるデータの形を定義していますが，このうちRBFタイプ用のstructの定義が曖昧です．それは，structのメンバーのsg__sallocとsg__attの間に4バイト分の未使用領域があるにもかかわらず，それを忘れているのです．ですから，この間にsg__null[4] とでもつけておいて下さい．このためファイルのアトリビュートが読みたくても読めなくて苦しんだことがあるのでぜひみなさんは注意して下さい．「自分のプログラムが動かないからコンパイラにバグがある」という考え方は危険ですので，まず自分を疑うようにしましょう．

グラフィック・パッケージ

OS-9/MICROWARE C上で使用可能

一般に，8ビットCPUで動作するOSは，汎用性を重視する目的で，画面表示はキャラクタ単位に限定している場合が大半です(CP/MやFLEXなど).

その中にあって，OS-9は，FMシリーズ用に限り，グラフィックをサポートしており，エスケープシーケンスもどきのキャラクタ列をコンソールに送るだけで実現できます．送出するパラメータは，通常アセンブラレベルで共有RAMを通してサブシステムに送出するデータと同じものなので，こうした事例を経験したことのない者にとっては，すぐ使えるというものではありませんが，いかなる言語からでもアセンブラを使うことなく応用できるという点は注目に値します．

ここでは,OS-9上で走るMICROWARE Cにおいてグラフィック処理を試みてみました(**リスト1**)．具体的には，コンソールに送るサブシステムコマンドを細かくかみくだいて，たとえばLINEなら，座標値等のパラメータを引数とするline(　)という関数を用意しました．

これにより，ことにC言語上におけるグラフィックの扱いは非常に容易なものとなります．

これまで，グラフィック機能を持ち合わせたコンパイラの多くは，数値計算には整数しか使えなかったため，特に3Dグラフィックを実現する場合には，座標計算がネックとなり，BASICインタプリタを使うことがほとんどでしたが，これに三角関数等の未サポート関数などを追加することによって，より高度なグラフィックが楽しめることでしょう．

■分割コンパイル

OS-9およびMICROWARE Cコンパイラは，どちらも多くのメモリを使うために，大きなプログラムを一度にコンパイルするのはむずかしいことです．ひとつの方法としては，OS-9のブート時のモジュールを最小限におさえ，でき

るだけ多くの作業領域を確保することが考えられますが，こうした方法に触れることは本書の目的ではないので，他の参考書を読んでください．

ここに紹介したグラフィックパッケージは，200行程度のものですが，これだけでも比較的大きなものです．よって，メインプログラムから呼ぶ場合もincludeするのではなく，分割コンパイルしておいてあとでリンクするという手法をとった方が安全です．具体的な使い方については後で述べます．

■各関数の仕様と使い方

cls(　)　引数なし　(画面消去)

プログラムを見てもわかるように，ただ単にwrite関数を使って，コントロールコード$0Cをコンソールに送っているだけです．当初，サブシステムコマンドの「ERASE」をパッチしてみましたが，ウィンドゥを複数にしたとき，いずれかの画面において，これが実行されるとすべてが消去されてしまうというトラブルが起こったために，この方法に変更しました．

さらに，なぜputchar(　)関数を使わなかったかというと，これはただ単に<stdio.h>をincludeしたくなかったというだけの理由です．

nline(a, b, c, d, fnc, col, bxf)　(直線)

直線を描く関数です．座標系は，BASICにおけるそれと全く同じで，640×200である．もちろん，色指定(col：0～7)や機能(fnc：0～4)，ボックス(bxf：0～3)も指定できますが，始点，終点の座標(a，d)－(c，d)は，必ず画面内の点でなければなりません．仮に，はみ出した点を引数として与えても，負数の場合は0，639(199)を超えた場合は639(199)として与えられたものとして処理します．他のパラメータが範囲外の場合も，特定の値に変更されます．各パラメータは次のような意味をもちます．

色指定　col：0～7

番　号	0	1	2	3	4	5	6	7
色	黒	青	赤	紫	緑	水色	黄	白

機能　　fnc：0～4

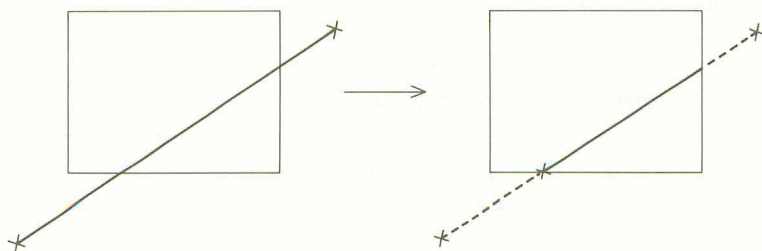
番　号	0	1	2	3	4
色	PSET	PRESET	OR	AND	XOR

line(p, q, r, s, fnc, col, bxf) (クリッピング機能付直線)

先の line(　)関数では，画面からはみ出た部分については無理矢理範囲内におさめてしまいました．あらかじめ範囲内にしか表示され得ないことがわかっているような場合には，これでも構いませんが，こと3Dグラフィックや入力データによるグラフ書きなどの場合には，いつどんなときに範囲外となるかもしれません．クリッピングというのは，ワールド座標系において仮想的に直線を引いたとき，ウィンドゥ内に見えるはずの直線を，範囲内の2点の座標で示すことです．

本プログラムでは，きわめて算術的にこの作業を行っているために，与えられた2点がどちらも範囲内にある場合でもIF文を何度も通るので必ずしもよいプログラムとはいえませんが，アルゴリズムには誤りはないので期待通りの動作はします．

なお，座標およびパラメータの当え方は nline(　)関数と全く同じです．

クリッピング

circle(xx, y, rx, ry, fnc, col) (円)

円の描き方には何通りかのやり方がありますが，ここでは32角形近似による方法をとっています．F-BASIC における円の描画は64角形近似ですから，若干見た目はよくありませんが，OS の関係上，64角形にすると速度が相当遅くなるので32角形としました．用意する三角関数データも半分ですみます．

プログラムはいたって簡単で，単位円のデータに横半径（rx）および縦半径（ry）を乗じて中心座標(x, y)に加えているだけです．また，正負を変えるだけで，点対称な座標が得られることから実際の描画は右左から上下へとのびていきます．色(col）および機能(fnc)については，nline(　)の場合と全く同じです．なお，プログラム中，4つの line(　)を，すべて nline(　)に直せば，F-BASICの円と同じに範囲外の部分がつぶれて表示されます．

pset(x, y, fnc, col)（点）

指定座標（x, y）に点を描く関数です．機能 (fnc) および色 (col) については，nline(　) と全く同じです．

paint(x, y, col, bc)（ペイント）

BASIC における PAINT とほとんど同じです．が境界色 (bc) の指定は混乱を避けるために1色としました．

symbol(x, y, fnc, col, arg, wd, hi, str)（シンボル）

BASIC における SYMBOL と全く同じです．文字の左上の座標を（x, y)，向き (arg) は 0～3，拡大幅 (wd) および拡大高 (hi) はそれぞれ1～255で指定します．文字列は，ポインタとして考えるので，次のようにします．

```
char  *S;
S="OS-9";
symbol (320, 100, 0, 7, 0, 2, 2, S);
```

■コンパイルの仕方とサンプルプログラム

グラフィック・パッケージはこれだけでは動きません．実際に動作させるためには，main(　)を含むファイルが別に必要です．

ここでは，グラフィックパッケージ中の関数をすべて使う簡単なサンプルプログラムを示します（リスト2)．

MICROWARE Cでは,複数のファイルを一括してコンパイルできるので(分割コンパイルしてリンクするという作業を自動的にする),ここではグラフィック・パッケージのファイル名をgraph. c，サンプルのファイル名をsample. c とでもして，次のように，

```
cc1 sample. c graph. c ↵
```

とするだけでコンパイルされます．なお，これに先立ち，cc1 のあるディレクトリを実行ディレクトリに，ソースファイルのあるディレクトリをワーキングディレクトにしておかなければなりません．

エラーなしにコンパイルが終了すれば，オブジェクトは cc1 のあるディレクトリにできます．ファイル名は，main のあるファイル名から拡張子（.c)を取り除いたものになりますから，コンパイル終了直後なら，たとえば，

sample ⏎

で実行されます．

■三角関数の追加とリサージュ図形

3D グラフィック等を実現するためにどうしても必要なのが，三角関数です．しかし，MICROWARE C では，三角関数はサポートしておらず，ユーザが独自に用意してやる必要があります．ここでは三角関数の例およびこれを応用したリサージュ図形のプログラムを示すことにします．

三角関数の作り方はいろいろありますが，多くの場合，テーブルを用意して補間します．テーブルを大きくとれば補間は簡単なもので済むので，プログラムは楽ですが，オブジェクトは大きくなります．逆にテーブルを小さくすると，補間プログラムは複雑になるかわりにオブジェクトは小さくなります．

ここでは，cos(　)を degree で0°～90°までテーブルとして用意し，補間は一次としましたので，プログラム的には前者の方法に近いといえます．補間が単純だけに速度は速くなっています．プログラムを**リスト3**に示します．

テーブルは dcos(　)という関数を通してアクセスされ，radian で与えられる引数から三角関数を与えるのは cos(　)と sin(　)で，sin(　)はただ単に cos(　)の位相を $\pi/2$ ずらしているだけです．

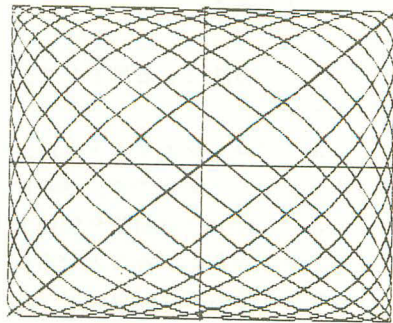
tan(　)は，sin(　)／cos(　)で与えられますが，cos(　)→0 のとき sin(　)→1，tan(　)＝∞となるので，|cos(　)|＜1×10^{-37}のときは tan(　)は1×10^{37}または-1×10^{37}を与えるようにしています．

リサージュ図形のプログラムを**リスト4**に示します．

このプログラムではグラフィック・パッケージの line(　) 関数と，三角関数パッケージの sin(　)関数を使うので，どちらも使用します．コンパイルするときは，リサージュのプログラムのファイル名を sample 2.c，三角関数のファイル名を sct.c とでもして

cc 1　sample 2. c　graph. c　sct. c⏎

とします．

リサージュ図形の出力例

■OS-9上でサブシステムを扱う方法

最初に述べたように，OS-9上からサブシステムを扱うには，コンソールへのキャラクタ送出だけで結構です．ここでは，そのキャラクタ列（コマンドパケットと呼ばれます）のフォーマットについて解説します．

コマンドパケットは，ヘッダ部，コマンド列，エンドコードの3つから成り立ちます．ヘッダ部は4バイトで，その先頭バイトは$14(DC4)とし，コマンドパケットの宣言をします．

ヘッダ

第0バイト　$14(DC4)
第1バイト　$0E(SO)　または　$0F(SI)
・$0E(SO)は，サブシステムへのコマンドの送出のみの場合(subout)
・$0F(SI)は，復帰情報を必要とする場合(subin)
の選択をします．
第2バイト　subin時の復帰情報のバイト数を示します．
第3バイト　subout時の送出バイト数を示します．

サブシステムコマンド例

共有RAMに書き込むデータの先頭から必要なバイト数のみ与えます．ここで与えるバイト数は，ヘッダの第3バイトの値と一致しなければなりません．

エンドコード

コマンドパケットの最後を宣言するためのもので，$03(EXT)を1バイト与えます．

次に，この方法による例として，コンソールの初期化プログラムを示します(リスト5)．

C言語でプログラムするときは，パラメータの与え方を簡略化するために，構造体を使うのが便利です．さらに，実際にコマンドパケットをコンソールに送るときは，write( )関数を使います．char( )を使うと，出力バッファを通るため，トラブルを生ずることがあります．

このプログラムでは，コンソールのカラーモードとグリーンモードを切り換えるためのもので，たとえば，console.cというファイル名でコンパイルした場

合次のようになります.

console　c　⏎　　(カラーモード)
console　g　⏎　　(グリーンモード)

グリーンモードはカラーモードに比べて，表示やスクロールが速いので，スクリーンエディタを使用したりするときには便利です.

なお，このプログラムでは，「画面消去せずに初期化する」機能を利用することによって，PFキーの内容表示を画面の最上段に置くなどということをしていますが，プログラムをよく読めばどうなっているかわかることと思います.

パレットの切り換え

グリーンモードにしたときでも，画面には文字を白で出したい場合があります.そこで,パレットの切り換えをするプログラムを作ってみました(**リスト6**).

プログラム中，palette(oc, nc)という関数は，直接I/Oをアクセスすることで，パレットコード(oc)にカラーコード(nc)を割り当てています.main( )は，コマンド入力時のパラメータ変換をするためのものです．ファイル名を，palette.cとしてコンパイルしたときは，

palette　4, >　　(コンマは空白でもよい)

で，パレットの4が白になります.

なお，当然のことながら，このプログラムは，FM-7, NEW7, 77でしか意味をもちません.

■Source List

リスト1

```
/*                                                  */
/*      Graphic Package     for FM-7/8              */
/*                                                  */
/*         by T.KANO      '84.9.17                  */

cls()
{
   char s[1];
   s[0] = 0x0c;
   write(1,s,1);
}

nline(a,b,c,d,fnc,col,bxf)
int a,b,c,d,fnc,col,bxf;
{
   static struct {
      char head[9];
      int point[4];
      char tail[2];
      } subline = {
         {0x14,0x0e,0,14,
          0, 0, 0x15, 7, 0},
         {0, 0, 0, 0},
         {0, 3}
         };

   a = (a<0) ? 0 : ((a>639) ? 639 : a)
   b = (b<0) ? 0 : ((b>199) ? 199 : b)
   c = (c<0) ? 0 : ((c>639) ? 639 : c)
   d = (d<0) ? 0 : ((d>199) ? 199 : d)
   fnc = (fnc<0 || fnc>4) ? 0 : fnc;
   col = (col<0 || col>7) ? 7 : col;
   bxf = (bxf<0 || bxf>2) ? 0 : bxf;

   subline.point[0] = a;
   subline.point[1] = b;
   subline.point[2] = c;
   subline.point[3] = d;
   subline.head[8] = fnc;
   subline.head[7] = col;
   subline.tail[0] = bxf;

   write(1,&subline,19);
}

#define swap(x,y) {float dumy; dumy=x; x=y; y=dumy;}

line(p,q,r,s,fnc,col,bxf)
int p,q,r,s,fnc,col,bxf;
{
   float a,b,c,d;

   a = p;
```

```
    b = q;
    c = r;
    d = s;

    if(a > c) {
       swap(a,c);
       swap(b,d);
       }
    if(a<=639.0 && c>=0.0) {
       if(a < 0.0 && a != c) {
          b = b + (d - b)*(-a)/(c - a);
          a = 0.0;
          }
       if(c > 639.0 && a !=c) {
          d = b + (d - b)*(639.0 - a)/(c - a);
          c = 639.0;
          }

       if(b > d) {
          swap(a,c);
          swap(b,d);
          }
       if(b<=199.0 && d>=0.0) {
          if(b < 0.0 && b != d) {
             a = a + (c - a)*(-b)/(d - b);
             b = 0.0;
             }
          if(d > 199.0 && b != d) {
             c = a + (c - a)*(199.0 - b)/(d - b);
             d = 199.0;
             }

             p = a;
             q = b;
             r = c;
             s = d;
             nline(p,q,r,s,fnc,col,bxf);
       }
    }
}

circle(x,y,rx,ry,fnc,col)
int x,y,rx,ry,fnc,col;
{
    static float sin[9] = {
    0.000000 , 0.195090 ,
    0.382683 , 0.555570 ,
    0.707107 , 0.831470 ,
    0.923880 , 0.980785 , 1.000000 };

    int i,x0,y0,x1,y1;

    x0 = rx;
    y0 = 0;

    for(i = 1;i < 9; i++ ){
```

```
        x1 = rx * sin[8-i];
        y1 = ry * sin[i];
        line(x+x0,y+y0,x+x1,y+y1,fnc,col,0);
        line(x+x0,y-y0,x+x1,y-y1,fnc,col,0);
        line(x-x0,y+y0,x-x1,y+y1,fnc,col,0);
        line(x-x0,y-y0,x-x1,y-y1,fnc,col,0);
        x0 = x1;
        y0 = y1;
        }
}

pset(x,y,fnc,col)
int x,y,fnc,col;
{
   if(x>=0 && x<=639 && y>=0 && y<=199){
      static struct {
      char head[8];
      int point[2];
      char tail[3];
      } subpset = {
         {0x14, 0x0e, 0, 10,
          0, 0, 0x17, 1},
         {0, 0},
         {0, 0, 3}
         };

      col = (col<0 || col>7) ? 7 : col;
      fnc = (fnc<0 || fnc>4) ? 0 : fnc;
      subpset.point[0] = x;
      subpset.point[1] = y;
      subpset.tail[0] = col;
      subpset.tail[1] = fnc;

      write(1,&subpset,15);

   }
}

paint(x,y,col,bc)
int x,y,col,bc;
{
   if(x>=0 && x<=639 && y>=0 && y<=199 ){
      static struct {
         char head[7];
         int point[2];
         char tail[4];
         } subpaint = {
         {0x14, 0x0e, 0, 10,
          0, 0, 0x18 },
         {0, 0},
         {0, 1, 0, 3}
         };

      subpaint.point[0] = x;
      subpaint.point[1] = y;
      subpaint.tail[0] = (col<0 || col>7 ) ? 7 :col;
```

```
            subpaint.tail[2]
             = ( bc >7 || bc<0 ) ? col : bc;
        write(1,&subpaint,15);
        }
}

symbol(x,y,fnc,col,arg,wd,hi,str)
int x,y,fnc,col,arg,wd,hi;
char str[];
{
   int i;
   for(i=0;i<=80 && str[i] !='\0';i++);
   if(i!=0 && x>=0 && x<=639 && y>=0 && y<=199) {
      static struct {
         char head[4];
         char cmd[8];
         int point[2];
         char cnt[1];
         } subsybl = {
         {0x14, 0x0e, 0, 0},
         {0, 0, 0x19, 0, 0, 0, 0, 0},
         {0, 0},
          0
         };
      char tail[1];

      subsybl.head[3] = 13 + i;
      subsybl.cmd[3] = col & 0x07;
      subsybl.cmd[4] = (fnc<0 || fnc>5) ? 0 : fnc;
      subsybl.cmd[5] = arg & 0x03;
      subsybl.cmd[6] = hi & 0xff;
      subsybl.cmd[7] = hi & 0xff;
      subsybl.point[0] = x;
      subsybl.point[1] = y;
      subsybl.cnt[0] = i;

      write(1,&subsybl,17);
      write(1,str,i);
      tail[0]= 3;
      write(1,tail,1);
      }
}
```

リスト2

```
/*                                      */
/*    Grapic Package test program       */
/*                                      */
/*      by T.KANO    '84.9.16           */

main()
{
   while(1) {
      circletest();
      painttest();
```

```
        symboltest();
        }
}

circletest()
{
    int y,ry;

    cls();
    for(ry=10;ry<=200;ry+=10)
        circle(320,100,ry*2,ry,0,7);
    cls();
    for(y=50;y<=150;y+=10)
        circle(y*4,y,200,100,0,7);
}
painttest()
{
    int i;

    cls();
    line(100, 50,250,100,0,1,1);
    line(100,120,250,190,0,2,1);
    line(300, 50,350,100,0,3,1);
    line(300,120,350,190,0,4,1);
    line(400, 50,600,100,0,5,1);
    line(400,120,600,190,0,6,1);

    for(i=1;i<8;i++){
        line(200, 70,500,170,0,i,1);
        paint(320,110,i,i);
        }

    for(i=-100;i<=100;i+=2){
        pset(320+i,100+i,0,2);
        pset(320-i,100+i,0,2);
        pset(320+i,100-i,0,5);
        pset(320-i,100-i,0,5);
        }
}

symboltest()
{
    int i;
    char *s;

    cls();
    s = "OS-9";
    for(i=0;i<14;i++)
        symbol(i*10,i*10,        0,(i%7)+1,0,1+i/4,1+i/4,s);
    for(i=0;i<14;i++)
        symbol(i*10,199-i*10,    0,(i%7)+1,1,1+i/4,1+i/4,s);
    for(i=0;i<14;i++)
        symbol(639-i*10,199-i*10,0,(i%7)+1,2,1+i/4,1+i/4,s);
    for(i=0;i<14;i++)
        symbol(639-i*10,i*10,    0,(i%7)+1,3,1+i/4,1+i/4,s);
}
```

リスト3

```
/*                                              */
/*   sin() cos() tan()  Package                 */
/*                                              */
/*     by T.KANO    '84.9.13                    */

float dcos(d)
int d;
{
    static float dat[91]=
    {        0, 0.017452, 0.034900, 0.052336, 0.069757,
     0.087156, 0.104528, 0.121869, 0.139173, 0.156434,
     0.173648, 0.190809, 0.207912, 0.224951, 0.241922,
     0.258819, 0.275637, 0.292372, 0.309017, 0.325568,
     0.342020, 0.358368, 0.374607, 0.390731, 0.406737,
     0.422618, 0.438371, 0.453991, 0.469472, 0.484810,
     0.500000, 0.515038, 0.529919, 0.544639, 0.559193,
     0.573576, 0.587785, 0.601815, 0.615661, 0.629320,
     0.642788, 0.656059, 0.669131, 0.681998, 0.694658,
     0.707107, 0.719340, 0.731354, 0.743145, 0.754710,
     0.766044, 0.777146, 0.788011, 0.798636, 0.809017,
     0.819152, 0.829038, 0.838671, 0.848048, 0.857167,
     0.866025, 0.874620, 0.882948, 0.891007, 0.898794,
     0.906308, 0.913545, 0.920505, 0.927184, 0.933580,
     0.939693, 0.945519, 0.951057, 0.956305, 0.961262,
     0.965926, 0.970296, 0.974370, 0.978148, 0.981627,
     0.984808, 0.987688, 0.990268, 0.992546, 0.994522,
     0.996195, 0.997564, 0.998630, 0.999391, 0.999848,
     1.000000 };

    int sign;

    d    = ( ( d < 0 ) ? -d : d ) % 360;
    sign = ( d > 179 ) ? -1 : 1;
    d %= 180;
    sign = ( d > 90 ) ? -sign : sign;
    d    = ( d > 90 ) ? 180 - d : d;

    return( sign * dat[ 90 - d ] );

}

float cos(r)
float r;
{   float dcos(),c;
    int d;

    r *= 57.2958;       /*  180/pai  */
    d = r;
    c = dcos(d);

    return( c + ( dcos(d+1) - c ) * ( r - d ) );
}

float sin(r)
float r;
```

```
{   float cos();

   return( cos( r - 1.570796));  /*   pai/2  */
}

float tan(r)
float r;
{   float t,sin(),cos();

   if( abs( t = cos(r) ) <= 1e-37 )
      if(sin(r) < 0.0 )
         return(-1e37);
      else
         return( 1e37);
   return( sin(r)/t );
}
```

リスト4

```
/*                      */
/*  Lissajous curves   */
/*                      */

#define STEP 0.05
main()
{
   int x0,y0,x1,y1;
   float c,r,sin();

   cls();
   line(320,  0,320,200,0,7,0);
   line(120,100,520,100,0,7,0);

   for( c = 0.0 ; c < 6.0 ; c += 0.5 ) {
      x0 = 320.0 + 200 * sin ( 0 );
      y0 = 100.0 - 100 * sin ( c );
      for( r = 0.0 + STEP ; r < 6.3 ; r += STEP ) {
         x1 = 320.0 + 200.0 * sin ( r );
         y1 = 100.0 - 100.0 * sin ( r + c );
         line(x0,y0,x1,y1,0,4,0);
         x0 = x1;
         y0 = y1;
         }
      }
}
```

リスト5

```
/*                      */
/*  console control     */
/*  by T.KANO '84.9     */
/*                      */

main(c,adr)
int c;
int *adr;
{
if(c == 2) {
   static struct {
      char head[4];
      char cmd[11];
      char tail[1];
      } cons = {
      {0x14,0x0e,0,11},
      {0,0,0x01,0,80,25,0,23,0xff,0xff,0},
      3
      };
   char *str;
   int i;

   str = adr[1];

   if(str[0] == 'C' || str[0] == 'c' ||
      str[0] == 'G' || str[0] == 'g' ) {
      cons.cmd[10] =
         (str[0] == 'C' || str[0] == 'c') ? 0 : 0xff;
      write(1,&cons,16);
      cons.cmd[7] = 25;
      cons.cmd[8] = 0;
      cons.cmd[9] = 0;
      write(1,&cons,16);
      str = "\n\012";
      for(i=0;i<47;i++)
         write(1,str,2);
      str = "\022\000\002";
      write(1,str,3);
      }
   }
}
```

リスト6

```
/*                      */
/*   change palette     */
/*                      */
/*   by T.KANO          */
/*         '84.9.18     */

main(n,ad)
int n;
int *ad;
```

```
{
   char *oc,*nc;

   if(n == 3){
      oc=ad[1];
      nc=ad[2];
         palette(oc[0]-'0', nc[0]-'0');
      }
}

palette(oc,nc)
int oc,nc;
{
   char *pl;

   pl = 0xfd38;
   if(oc>=0 && oc<=7 && nc>=0 && nc<=7)
      pl[oc] = nc;
}
```

各処理系の関数一覧表

各処理系における，言語機能，UNIXとの互換性などの確認のためにこの表を作成しました．ただし，C言語は言語機能がそれほど大きくないとはいえ，すべての機能について作動させて確認することは極めて困難なため，基本的にはマニュアルや参考文献に基づいて記述しました．

あくまでも1つの目安として利用してください．誤まりのある場合には御指摘いただきたいと思います．

一覧表の見方

一覧表の記号の意味は次のとおりです．

○：実際に作動を確認したもの．もしくはマニュアルに明記されているもの
×：実際に作動しなかったもの．もしくは作動しないことが明らかなもの
△：類似する関数を持つもので作動が確認されているもの
▲：類似する関数を持つもので作動が確認されていないもの

UNIXの欄のSYSはシステムコール，RUNはランタイム・ルーチンを表しています．なお，調査したシステムのバージョンは下記のとおりです．

OPTIMIZING C	ver. 2.10
Lattice C	ver. 2.0
DeSmet C	ver. 2.2
BDS C	ver. 1.5a
MICROWARE C	ver. 1.1.4
Aztec CII	ver. 1.06

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
#asm.endas			×			○	○
#define		○	○		○	○	○
#else		○			○		○
#endif		○			○	○	○
#if		○	○		○		○
#ifdef		○	○		○	○	○
#ifndef		○	○		○	○	○
#include		○	○		○	○	○
#line			○				
#undef		○	○		○		○
IBMPCks				○			
_inbuf						○	
_memory				○			
_more				○			
_os9						○	
_setsp				○			
_showsp				○			
abort	RUN	○				○	
abs	RUN			○	○	○	
access	SYS					○	
acos	RUN	○					○
alarm	RUN						
allmem			○				
alloc		○			○		
asin	RUN	○					○
assert	RUN						

各処理系の関数一覧表

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
atan	RUN	○					○
atan2	RUN	○					○
atof	RUN	○	○			○	○
atoi	RUN	○	▲	○	○	○	○
atol	RUN			○		○	○
auto		○	○		○	○	○
bdos		○		○	○		○
big		○					
bios					○		○
biosh					○		△
brk	SYS						
calloc	RUN	○	○	○		○	○
ccall					▲		
ccalla					▲		
ceil	RUN	○					○
cfsize					○		
cgets			○				
chdir	RUN	○				○	
chmod	SYS	○				○	
chown	SYS						
chxdir						○	
ci				○			
clearerr	RUN	○	▲			○	○
close	SYS	○	○	○	○	○	○
clrplot					○		
co				○			

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec CII
codend					○		
compact			○				
coreleft		○					
cos	RUN	○		○		○	○
cprintf			○				
cputs			○				
crc						○	
create	SYS	○	○	○	○	○	○
crypt	RUN						
cs				○			
cscanf			○				
csts				○			
csw					○		
defdrive						○	
direct						○	
ds				▲			
dup	SYS					○	
encrypt	RUN						
endext					○		
errmsg					○		
errno	SYS				○		
exec	RUN				○	▲	
execl	RUN				○		○
execv					○		○
exit	RUN	○	○	○	○	○	○
exp		○		○			○

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
exp10				○			
extern		○	○		○	○	○
externs					○		
fabort					○		
fabs		○		○			○
farcall		○					
fcbaddr					○		
fclose	RUN	○	○	○	○	○	○
fcreate					○		
fdopen	RUN						○
feof	RUN	○	○			○	○
ferrer	RUN	○	○			○	○
fflush	RUN	○	○		○	○	○
fgetc	RUN	○	○	○			
fgets	RUN	○	○	○	○	○	○
fileno		○	○			○	○
findndtr						○	
findstr						○	
floor	RUN	○					○
fopen	RUN	○	○	○	○	○	○
fork	RUN					▲	
formal			○				
fprintf	RUN	○	○	○	○	○	○
fputc	RUN	○	○	○			
fputs	RUN	○	○	○	○	○	○
frand				○			

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
fread	RUN	○		○		○	○
free	RUN	○	○	○		○	○
freeall				○			
freemem						○	
freopen	RUN		○				○
fscanf	RUN	○	○	○	○	○	○
fseek	RUN	○	○	○		○	○
ftell	RUN	○	○			○	○
ftoa		○					○
fwrite	RUN	○		○		○	○
gcd	RUN						
getc	RUN	○	○	○	○	○	○
getch			○		○		
getchar	RUN	○	○	○	○	○	○
getenv	RUN						
getgrent	RUN						
getime						○	
getlin					▲		
getlogin	RUN						
getmem			○				
getpid	▲					○	
getpw	RUN						
getpwent	RUN						
gets	RUN	○	○	○	○	○	○
getstat						○	
getuid						○	

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec CII
getval					○		
getw	RUN	○		○	○		○
index	RUN	○		○		○	○
initb					○		
initw					○		
inp		▲			○		△
inp16		▲					
intrinit		○					
isalnum	RUN	○	○	○		○	○
isalpha	RUN	○	○	○	○	○	○
isascii	RUN	○	○	○		○	○
isatty	RUN						○
iscntrl	RUN	○	○	○		○	○
iscsynf			○				
iscsynm			○				
isdigit	RUN	○	○	○	○	○	○
isgraph			○				
islower	RUN	○	○	○	○	○	○
isprint	RUN	○	○	○		○	○
ispunct	RUN	○	○	○		○	○
isspace	RUN	○	○	○	○	○	○
isupper	RUN	○	○	○	○	○	○
iswap		○					
isxdigit			○				
itoa		○	▲				
itob		○					

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
itom	RUN						
kbhit					○		
kill	RUN					○	
13tol	RUN					○	
line					○		
link	SYS						
lmove				○			
lock	SYS						
log	RUN	○		○			○
log10	RUN	○		○			○
longjmp	RUN	○		○	○	○	○
lower		○					
lseek	SYS	○	○	○		○	○
ltell		○					
lto13	RUN					○	
ltoa		○					
ltoh		○					
madd	RUN						
makefcb		○					
malloc	RUN	○	○	○		○	○
max					○		
mdiv	RUN						
medium							
min	RUN				○		
mknod						○	
mktemp	RUN					○	

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
modf	RUN	○					○
modlink						○	
modload						○	
monitor	RUN						
movblock		○					
movemem		○	○		○		○
msub	RUN						
mult	RUN						
munlink						○	
nice	RUN						
nlist	RUN						
nrand					○		
oflow					○		
open	SYS	○	○	○	○	○	○
outp		○			○		△
outp16		○					
pause	RUN				○	○	
pclose	RUN						
peek		○			○		
perror	RUN					○	
pkclose	RUN						
pkfail	RUN						
pkopen	RUN						
pkread	RUN						
pkwrite	RUN						
plot	RUN				○		

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
poke		▲			○		
popen	RUN						
pow	RUN	○		○			○
printf	RUN	○	○	○	○	○	○
ptrace	RUN						
putc	RUN	○	○	○		○	○
putch			○		○		
putchar	RUN	○	○	○	○	○	○
puts	RUN	○	○	○	○	○	○
putw	RUN	○		○	○	○	○
qsort	RUN	○		○	○	○	○
rand	RUN			○	○		△
rbrk			○				
read	SYS	○	○	○	○	○	○
readln						○	
realloc	RUN	○		○			○
register		○	×		×	○	○
rename		○		○	○		○
repmem			○				
rewind	RUN		○	○		○	
rindex	RUN	○		○			○
rlsmem			○				
rpow	RUN						
rstmem			○				
rsvstk					○		
rtell						○	

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
sbrk	SYS	○	○		○	○	○
scanf	RUN	○	○	○	○	○	○
seek					○		
segread		○					
setbuf	RUN					○	○
setfcb					○		△
setime						○	
setjmp		○		○	○	○	○
setmem		○	○	○	○		○
setnbf			○				
setplot					○		
setpr						○	
setstat						○	
setuid	RUN					○	
signal	RUN					○	
sin	RUN	○		○			○
sitmem			○				
sleep	RUN				○	○	
small		○	○				
sprintf	RUN	○	○	○	○	○	○
sqrt	RUN	○		○			○
srand	RUN			○	○		
srand1					○		
sscanf	RUN	○	○	○	○	○	○
stacksize						○	
static		○	○		×	○	○

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
stcarg			○				
stch _ i			○				
stci _ d			○				
stcis			○				
stcisn			○				
stclen			○				
stcmp			○				
stcmpa			○				
stcu _ d			○				
stpblk			○				
stpbrk			○				
stpchr			○				
stpsym			○				
stptok			○				
strcat	RUN	○	○	○	○	○	○
strcmp	RUN	○	○	○	○	○	○
strcpy	RUN	○	○	○	○	○	○
strhcpy						○	
strlen	RUN	○	○	○	○	○	○
strncat	RUN	○		○		○	○
strncmp	RUN	○		○		○	○
strncpy	RUN	○	▲	○		○	○
stscmp			○				
stspfp			○				
swab	RUN						
swapin					○		

	UNIX Ver.7	OPT. C86	Lattice C	DeSmet C	BDS C	MICRO WARE C	Aztec C II
sync	RUN						
sysint		○					
system	RUN	○				○	
tan		○		○			○
tell	SYS				○		
textplot					○		
time	RUN						
times	RUN						
tolower		○	○	○	○	○	○
topofmem			○	○	○		
toupper		○			○	○	○
tsleep						○	
ttyname	RUN						
ttyslot	RUN						
ungetc	RUN	○	○		▲		○
ungetch		○	○				
unlink	SYS	○	○	○	▲	○	○
utoa		○			○	○	
wait	RUN					○	
wqsort		○					
write	SYS	○	○	○	○	○	○
writeln						○	

参考文献

〔Cに関するもの〕

- 『プログラミング言語C』 B.W.カーニハン，D.M.リッチー （石田晴久訳），共立出版
- 『インターフェース』 '84／6 「Cのすべて」，CQ出版社
- 『インターフェース』 '84／7 「続・Cのすべて」，CQ出版社
- 『電子科学』 '84／1 「これからの標準プログラミング言語C」，廣済堂産報出版
- 『電子科学』 '84／2 「各種コンパイラの徹底的機能比較」 宇津木真，廣済堂産報出版
- 『マイコンピュータ』 第10号 「C言語の研究」，CQ出版社

〔UNIXに関するもの〕

- 『UNIXビギナーズ・ガイド』 高玉皓司，ダムラン
- 『UNIXシステム入門 I』 H.マギルトン，R.モーガン （玄光男 荒実共訳），マグロウヒルブック
- 『UNIX入門』 R.トーマス，G.イエーツ （I/O編集部訳），工学社

〔その他〕

- 『ソフトウェア作法』 B.W.カーニハン，P.J.プローガー （木村泉訳），共立出版
- 各種CPU，処理系，OS等のマニュアル．

万一ご質問等がありましたら，文書で小社第2編集部宛お寄せ下さい．

Cサンプル&ツール集

昭和59年10月25日　初版第1刷発行
昭和61年5月10日　初版第4刷発行

著　者　C言語研究会
発行者　片岡　巖
発行所　株式会社 技術評論社
東京都千代田区九段南2-4-13
電話　03(262)9351（営業部）
03(237)8315（編集部）
印　刷　萬友社
製　本　村上製本所

定価はカバーに表示してあります

ISBN4-87408-150-9 C3054

表紙デザイン　小倉一夫